EasyDelphi: Introdução a POO com Delphi desde o mais básico até o desenvolvimento avançado

Clube Delphi +PHP

DevMedia group

Ano 9 :: Edição 109 :: R$ 11,90

Ask the Expert
InputBox com ComboBox

QuickUpdate
Delphi versus Robótica

Banco de Dados
Veja na prática como usar PostGreSQL com RAD Studio 2009

Expert
Crie objetos auto-instanciáveis usando Orientação a Objetos

PHP
Trabalhando com arquivos XML na prática com o PHP

Aplicações Multi-Banco com ZeosDBO

Aplicações multi-banco sem complicação

Na Web

Confira no portal ClubeDelphi PLUS 3 vídeo-aulas:

- Curso Delphi Prism: Aulas de 1 a 3

Novidades/ Tutorial
Domine o Delphi Prism e crie aplicações robustas com ele

Novidades/ Web
Exiba relatórios na Web usando RAVE Reports e ASP.NET

ISSN 1517990-7

9 771517 990009 001109

São Paulo Tel: (11) 3717-1331 ou (11) 3717-5231
Rio de Janeiro Tel: (21) 3879-5400 ou Fax: (21) 3879-5407
Minas Gerais Tel: (31) 3231-4331
Santa Catarina Tel: (47) 3001-5131
Porto Alegre Tel: (51) 3251-4331

9h às 18h - horário de Brasília de segunda à sexta-feira, sábados 9h às 12h.

RJHOST

PROMOÇÃO
DOMÍNIO GRÁTIS [2]

Assine o Plano Básico por 1 ano ou os planos Avançado e Profissional por 6 meses e ganhe 1 domínio internacional[1]

SITE PRONTO GRÁTIS EM TODOS OS PLANOS

SISFÁCIL

COMPACTO
- HTML
- FLASH
- PHP*
- ASP*
- BANCO DE DADOS *
- CPANEL *
- FANTÁSTICO *
- PLESK *

A PARTIR R$ 6,90

BÁSICO
- HTML
- FLASH
- PHP*
- ASP*
- BANCO DE DADOS *
- CPANEL *
- FANTÁSTICO *
- PLESK *

A PARTIR R$ 13,90

AVANÇADO
- HTML
- FLASH
- PHP*
- ASP*
- BANCO DE DADOS *
- CPANEL *
- FANTÁSTICO *
- PLESK *

A PARTIR R$ 24,90

PROFISSIONAL
- HTML
- FLASH
- PHP*
- ASP*
- BANCO DE DADOS *
- CPANEL *
- FANTÁSTICO *
- PLESK *

A PARTIR R$ 34,90

*RECURSOS DE ACORDO COM O PLANO ESCOLHIDO

NA RJHOST VOCÊ ESCOLHE E GANHA

DOBRO ESPAÇO EM DISCO

ou

DOBRO TRÁFEGO MENSAL

WWW.RJHOST.COM.BR

1 - Parceria com o site:www.e-registros.com.br 2 somente os domínios que terminam com .com, .net, .biz, .info, .org que participam desta promoção.

Sumário

Olá, eu sou o **DevMan**! Desta página em diante, eu estarei lhe ajudando a compreender com ainda mais facilidade o conteúdo desta edição. Será um prazer contar com sua companhia! Confira abaixo o que teremos nesta revista:

Delphi

Delphi Core

10 – Conectividade com ZeosDBO
Criando aplicações multi-banco com o ZeosDBO
[*Daniel Sobrinho*]

Banco de Dados

18 – Delphi e PostGreSQL
Como desenvolver aplicações utilizando o banco de dados free PostGreSQL?
[*Adriano Santos*]

Expert

23 – POO: Objetos Auto-Instanciáveis
Utilizando cláusulas opcionais da Unit para criar objetos dinamicamente
[*André Dias Martins e Marcos Aurélio Rocha*]

Novidades/ Tutorial

27– Curso de Delphi Prism – Parte 2
Novidades da Linguagem – Métodos Anônimos e Delegates
[*Guinther Pauli*]

Novidades/ Web

33 – Solução para relatórios WEB
Integre sua aplicação com Web Service e crie relatórios com o Rave Reports
[*Maikel Scheid*]

Easy Delphi

Easy Delphi

40 – Introdução à POO
Aprenda, de uma forma simples, a programar da forma correta
[*Guinther Pauli*]

PHP

PHP

45 – Trabalhando com XML no PHP
Aprenda a mapinular estes tipos de arquivos com classes nativas do PHP
[*Rodrigo Carreiro*]

Você percebeu os ícones ao lado de cada matéria? Eles indicam o que você vai encontrar no artigo – dessa forma, você também pode ter uma idéia geral do que vai encontrar nesta edição como um todo! Os editores trabalham sempre no sentido de fechar a revista seguindo esta definição, para oferecer a você o melhor conteúdo didático!

Confira abaixo a lista com a definição dos tipos de artigo encontrados nesta edição:

[Boa Idéia] Esse tipo de artigo é quase um artigo "Mão na Massa" - a diferença é que o exemplo do artigo é tão criativo que é considerado uma 'boa idéia', dentro do contexto de desenvolvimento de software.

[Expert] Artigo com foco no leitor avançado.

[Novidades] Artigos sobre tecnologias de ponta, que ainda não fazem parte o dia-a-dia do desenvolvedor mas que prometem ser a próxima febre.

[Easy Delphi] Artigo com foco no leitor iniciante.

[PHP] Artigo com foco no PHP puro ou Delphi for PHP.

Clube Delphi + PHP

Ano 9 - 109ª Edição 2009 - ISSN 1517990-7 Impresso no Brasil

Corpo Editorial

Editor Geral
Adriano Santos
adriano.santos@devmedia.com.br

Editor Técnico
Paulo Quicoli
pauloquicoli@devmedia.com.br

Daniel Sobrinho -
daniel.sobrinho@devmedia.com.br

Comissão Editorial
Fabrício Desbessel, Paulo Quicoli, Adriano Santos e Guinther Pauli.

Capa
Romulo Araujo - romulo@devmedia.com.br

Diagramação
Gabriela de Freitas - gabrieladefreitas@gmail.com

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Distribuição
Fernando Chinaglia Dist. S/A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Cristiany Queiroz – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 3382-5038

Kaline Dolabella – Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!
Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Adriano Santos - Editor da Revista
adriano.santos@devmedia.com.br

EDITORIAL

Olá amigos, chegamos a edição 109 da revista ClubeDelphi e é sempre um prazer fechar cada edição, ainda mais quando temos artigos de temas tão interessantes. Na capa desse mês trouxemos um tema bastante procurado por todos: Sistemas Multi-Banco. São poucas as suítes de componentes de acesso a dados que nos possibilitam desenvolver sistemas com esse comportamento. Por isso mostraremos nessa edição a suíte ZeosDBO. O nosso editor técnico Daniel Sobrinho fez uma análise profunda dos componentes e decidiu nos falar um pouco sobre eles. É o que veremos nas próximas páginas.

Na sequência vem nosso amigo Guinther Pauli com a continuação dos seus cursos de Delphi Prism e Orientação a Objetos. Em Delphi Prism aprenderemos um pouco mais sobre a nova linguagem, carinhosamente chama de D# pelo autor. Guinther detalha sobre métodos anônimos e Delegates. De fato o artigo está muito bom. Ele ainda fala mais sobre Orientação a Objetos na seção EasyDelphi, dessa vez falando de Polimorfismo, um tema importantíssimo para quem quer ingressar nesse mundo.

Falando em POO, o André e o Marcos mostram como desenvolver objetos independentes e que conseguem ser auto-instanciáveis. Verdade, o artigo está bem legal e mostra um lado da orientação a objetos que pouca gente conhece. Vale a pena conferir.

E que tal experimentar outros bancos de dados Free? Gostou da idéia? Então veja uma pequena introdução ao PostGreSQL. Eu fiz questão de ver de perto esse SGBD e constatei que ele é muito robusto e inteligente. No artigo veremos como fazer sua instalação, download de sua suíte de componentes, criação do banco e utilização dele via Delphi.

Na seção Web temos dois excelentes artigos. O Maikel fala do desenvolvimento de relatórios em formato PDF utilizando o RAVE Reports no Delphi. Confesso que o artigo me surpreendeu, pois mostra algumas técnicas bem interessantes. Uma delas é o desenvolvimento de um Web Services capaz de gerar o relatório e devolver a URL dele ao sistema chamador, que por sua vez exibe o relatório. Muito bom por sinal. O Rodrigo fala específicamente sobre arquivos XML no PHP. Como gerar arquivos XML? Qual o formato padrão? Como ler esse tipo de arquivo? Enfim, todos os detalhes disso você confere com ele.

Espero que tenham gostado da pauta e aguardem porque mais novidades estão por vir.

Uma boa leitura, muito sucesso a todos e até a próxima.

Adriano Santos

adriano.santos@devmedia.com.br

Editor Chefe da revista Clube Delphi
Conheça nosso BLOG em *clubedelphiphp.blogspot.com*

ClubeDelphi PLUS

DevMedia group

A revista ClubeDelphi é parte integrante da assinatura ClubeDelphi PLUS.
Para mais informações sobre o pacote PLUS, acesse:
http://www.devmedia.com.br/clubedelphi/portal.asp

Hospede seus sites de forma simples, rápida e ilimitada.

Plano **Básico**

- 1 domínio principal
- 50 domínios hospedados
- Transferência **ilimitada**
- Espaço **ilimitado**
- 10 contas de e-mail
- PHP, ASP e ASP.NET
- Painel de controle Plesk
- Suporte 24hs

Tudo isso por apenas

R$ **15,90** por mês
pagamento trimestral

Assine agora mesmo e descubra por que Hospedix é a hospedagem de sites preferida entre os profissionais.

- Conheça também nossos planos profissionais com registro de domínio grátis e isenção de taxas anuais enquanto for cliente.

HOSPEDIX
HOSPEDAGEM PROFISSIONAL
www.hospedix.com.br

30 DIAS GRÁTIS

Ao assinar, digite o código de desconto exclusivo para leitores desta revista e ganhe 30 dias inteiramente grátis.

RVSTMEDIA

ClubeDelphi PLUS

Informativo ClubeDelphi

Portal ClubeDelphi

www.clubedelphi.net/portal

+ 700 vídeo aulas e 19 cursos online

Brinde na web desta edição - 3 vídeos! Confira:

3 Vídeos

• Curso de Delphi Prism com Guinther Pauli

Aprenda desde o início a desenvolver aplicações Web utilizando o novíssimo e simpático Delphi Prism. Guinther Pauli mostra todos os detalhes da linguagem, mudanças, particularidades etc.

http://www.devmedia.com.br/articles/viewcomp.asp?comp=13197
http://www.devmedia.com.br/articles/viewcomp.asp?comp=13198
http://www.devmedia.com.br/articles/viewcomp.asp?comp=13207

Para acessar os vídeos utilize os seguintes dados

Login: ***DVM.PLS*** *Senha:* ***406462***

Gostou das vídeo aulas? O portal *www.devmedia.com.br* possui mais de **2 mil vídeo aulas** e **dezenas de cursos online** sobre desenvolvimento de software! Agora você pode comprar as vídeo aulas que preferir e fazer sua própria combinação de vídeos! Saiba mais em *www.devmedia.com.br/creditos*

Edições Anteriores da ClubeDelphi

Você pode comprar todas as edições anteriores da ClubeDelphi através do site da DevMedia!

Para isso basta acessar *https://seguro.devmedia.com.br/edicoes_anteriores.asp*

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Para votar, você vai precisar do código e senha de banca desta edição, que é:
Login: **DVM.PLS** Senha: **406462**

Perguntas e Respostas

InputBox com ComboBox

Perguntas respondidas por Paulo Quicoli
pauloquicoli@devmedia.com.br

Como sempre digo: Quem procura acha. Durante o fechamento dessa edição recebi um e-mail de um colega pedindo uma ajuda. Ele queria montar um InputBox que tivesse um ComboBox dentro com valores fixos. Fiz uma pequena busca na internet e encontrei inúmeras funções, uma delas resolvi publicar para ficar de registro (**Listagem 1**). O código é até um pouco extenso eu confesso, mas é bem simples de entender. Rotinas como InputQuery, InputBox e até o MessageDlg são fáceis de personalizar, pois elas fazem uso de funções prontas no Delphi. Nesse caso é ainda mais fácil. Se analisarmos o InputQuery e InputBox veremos que os procedimentos na verdade apenas criam um form em tempo de execução e retornam o botão que foi clicado, apenas isso. E é o que fazemos na **Listagem 1**, confira. Forte abraço. ●

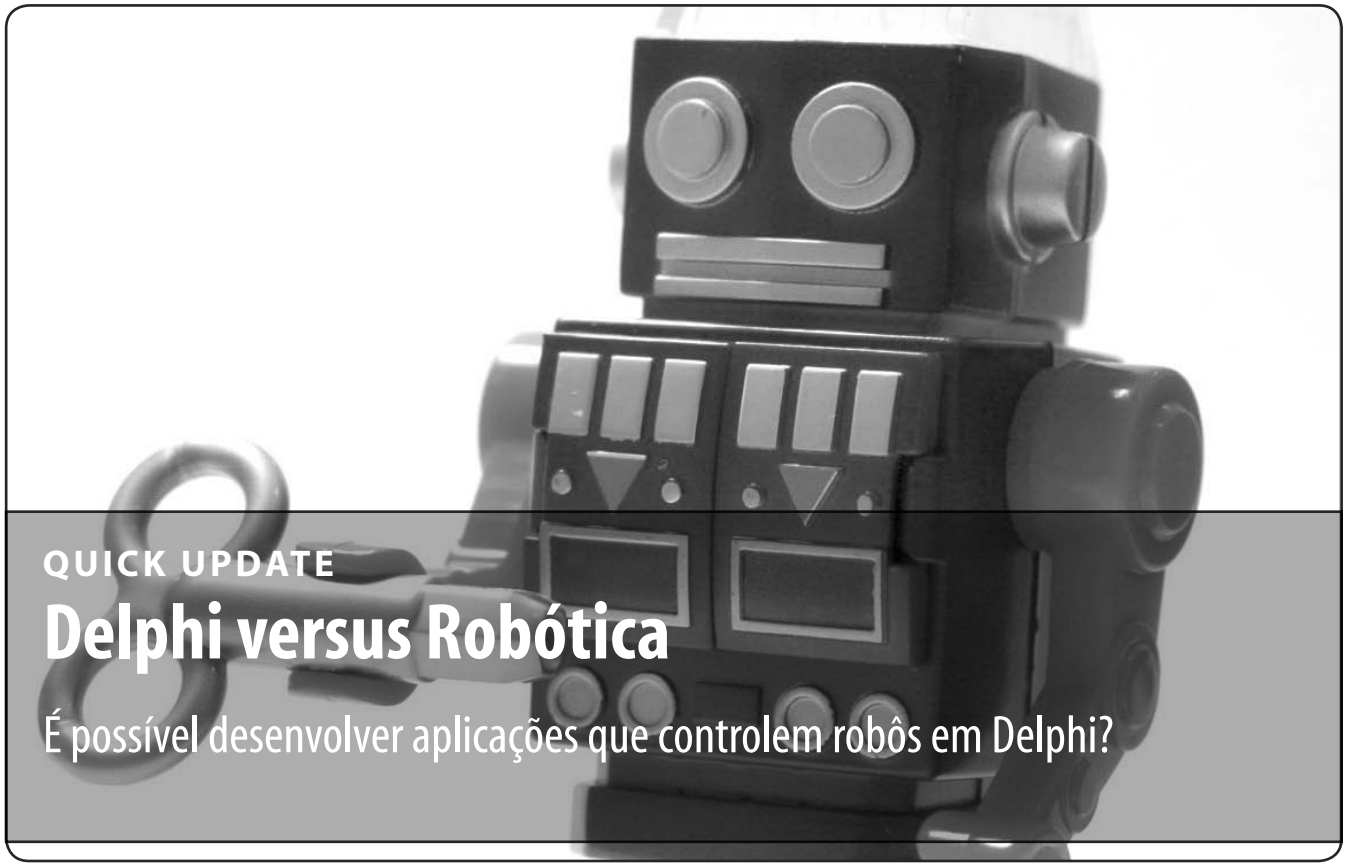
Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:
www.devmedia.com.br/clubedelphi/feedback

Listagem 1. Criando o InputBox

```
function InputCombo(const ACaption, APrompt: string; const AList:
TStrings): string;
  function GetCharSize(Canvas: TCanvas): TPoint;
  var
    I: Integer;
    Buffer: array[0..51] of Char;
  begin
    for I := 0 to 25 do Buffer[I] := Chr(I + Ord('A'));
    for I := 0 to 25 do Buffer[I + 26] := Chr(I + Ord('a'));
    GetTextExtentPoint(Canvas.Handle, Buffer, 52, TSize(Result));
    Result.X := Result.X div 52;
  end;
var
    Form: TForm;
    Prompt: TLabel;
    Combo: TComboBox;
    DialogUnits: TPoint;
    ButtonTop, ButtonWidth, ButtonHeight: Integer;
begin
    Result := '';
    Form   := TForm.Create(Application);
    with Form do
    try
      Canvas.Font := Font;
      DialogUnits := GetCharSize(Canvas);
      BorderStyle := bsDialog;
      Caption     := ACaption;
      ClientWidth := MulDiv(180, DialogUnits.X, 4);
      Position    := poScreenCenter;
      Prompt      := TLabel.Create(Form);
      with Prompt do
      begin
        Parent   := Form;
        Caption  := APrompt;
        Left     := MulDiv(8, DialogUnits.X, 4);
        Top      := MulDiv(8, DialogUnits.Y, 8);
        Constraints.MaxWidth := MulDiv(164, DialogUnits.X, 4);
        WordWrap := True;
      end;
      Combo := TComboBox.Create(Form);
      with Combo do
      begin
        Parent := Form;
        Style  := csDropDownList;
        Items.Assign(AList);
        ItemIndex := 0;
        Left      := Prompt.Left;
        Top       := Prompt.Top + Prompt.Height + 5;
        Width     := MulDiv(164, DialogUnits.X, 4);
      end;
      ButtonTop    := Combo.Top + Combo.Height + 15;
      ButtonWidth  := MulDiv(50, DialogUnits.X, 4);
      ButtonHeight := MulDiv(14, DialogUnits.Y, 8);
      with TButton.Create(Form) do
      begin
        Parent      := Form;
        Caption     := 'OK';
        ModalResult := mrOk;
        default     := True;
        SetBounds(MulDiv(38, DialogUnits.X, 4), ButtonTop,
         ButtonWidth,
          ButtonHeight);
      end;
      with TButton.Create(Form) do
      begin
        Parent      := Form;
        Caption     := 'Abbrechen';
        ModalResult := mrCancel;
        Cancel      := True;
        SetBounds(MulDiv(92, DialogUnits.X, 4), Combo.Top +
         Combo.Height + 1
          5, ButtonWidth, ButtonHeight);
        Form.ClientHeight := Top + Height + 13;
      end;
      if ShowModal = mrOk then
      begin
        Result := Combo.Text;
      end;
    finally
      Form.Free;
    end;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  List: TStringList;
begin
  List := TStringList.Create;
  try
    List.Add('Auswahl1');
    List.Add('Auswahl2');
    List.Add('Auswahl3');
    Label1.Caption := InputCombo('Input Combo', 'Caption', List);
  finally
    List.Free;
  end;
end;
```



QUICK UPDATE

Delphi versus Robótica

É possível desenvolver aplicações que controlem robôs em Delphi?

Adriano Santos

falecom@adrianosantos.pro.br

É desenvolvedor Delphi há mais de 10 anos, programador PHP, especialista em aplicações Win32 e API, bacharel em Comunicação Social. É Editor Chefe das revistas ClubeDelphi e WebMobile (Seção .net). Gerente de Desenvolvimento na SoftPark, membro fundador do DUG-SP (www.dug-sp.com) e mantenedor do blog Delphi to Delphi (www.delphitodelphi.blogspot.com).

É sabido de muitos desenvolvedores mais experientes que desenvolver aplicações que conversem em baixo nível com hardware só é possível com linguagens que tenham esse suporte, como C por exemplo. O desenvolvimento de driver por exemplo precisa de nível ring 0 para que se tenha acesso total ao hardware, portas, etc. O Delphi não possui esse acesso, mas mesmo assim é possível criar boas aplicações.

Há alguns meses a empresa que trabalho realizou um workshop onde mostramos nossas tecnologias. Antes desse evento acabamos conhecendo (minha equipe e eu) uma equipe bem interessante que trabalha com educação tecnológica através de robôs. Curiosos, acabamos chamando os colegas para demonstrar seus equipamentos e tivemos muitas surpresas. A empresa PETe (Educação Tecnológica) desenvolve robôs que são capazes de realizar diversas tarefas, tais como seguir luzes, reconhecer cores, fugir do escuro, levantar objetos etc. O mais interessante nisso tudo é que o software utilizado para controlar os robôs foi concebido em Delphi, isso mesmo, Delphi!

O software consiste em uma pequena IDE que possibilita programar os movimentos e ações do robô. A linguagem é proprietária e chama-se LEGAL. Foi desenvolvida pela PNCA, uma empresa do grupo. Com simples comandos a criança ou adulto consegue fazer os robôs girarem, seguirem a luz, fugirem do escuro, seguirem linhas e etc.

A empresa PETe trabalha em parceria com as secretarias de educação de municípios em São Paulo, Minas Gerais e outros estados. É vendido um kit de desenvolvimento somado as equipamentos necessários para montagem dos robôs, tais como braços mecânicos, leds, sensores etc. Um treinamento aos professores da rede de ensino também é disponibilizado, dessa forma os professores podem transmitir o conhecimento adquirido para seus alunos.

Além de escolas, também é possível adquirir os kits como pessoa fisica. O que mais me chamou atenção no projeto é que a IDE para programação do robô foi desenvolvida em Delphi, como já citei. A linguagem é simples, como no exemplo a seguir:

```
SE CONDICAO ENTAO
...//Executa comando
FIM
```

O intuito é ser o mais simples possível para que crianças aprendam rápido noções de lógica, semântica, etc. Na **Figura 1** vemos um dos robôs em ação em uma feira de tecnologia. Quem tiver interesse em ver como funciona, basta acessar os links no final desse artigo que o redirecionará para a página do YouTube, onde diversos vídeos estão disponíveis.

Um forte abraço a todos e até a próxima. ●

Figura 1. Robô em ação

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Links para os vídeos

FEIRA CIEE 2008

Participação da PNCA Robótica e Eletrônica apoiando a Feira do Estudante EXPO CIEE 2008, parque do Ibirapuera em São Paulo. O robô tem que remover de dentro da arena todos os objetos, mas ele não pode sair fora da área marcada.

http://www.youtube.com/watch?v=ACmxOqmp3LU

Capacitação de professores 2007

Capacitação de professores para o projeto nas escolas municipais de ensino fundamental em João Pessoa na Paraíba.

http://www.youtube.com/watch?v=xVI-jfZZuiY

GuindasteSBPC

Apresentação de uma aplicação do Kit ALFA da PNCA em parceria com o NIED da UNICAMP, durante o SBPC Jovem 2008.

http://www.youtube.com/watch?v=hlecqXRV10U

Agv eesc-usp 2006

Robô AGV vencedor do I SMART CHALLENGE na SEMATRON 2006 EESC-USP. Construído com o Kit ALFA da PNCA Robótica e Eletrônica.

http://www.youtube.com/watch?v=dfkc7VETgNQ

Aula ensino fundamental

Aula de robótica em escola municipal de ensino fundamental na cidade de Lins no estado de São Paulo. Atividade desenvolvida em parceria com o Centro Universitário de Lins – UNILIS, 2006.

http://www.youtube.com/watch?v=5MtezP4NR7c

AGV controlado por cores

O Agv construído segue a pista amarela para descarregar a caixinha da mesma cor, retornando depois para a região vermelha, onde esperará por nova carga.

http://www.youtube.com/watch?v=m9SgZHq5QqI

SEMATRON 2007

III SEMATRON 2007 - USP/ São Carlos - II SMART CHALLENGE - Veja o que é possível fazer em poucas horas... Parabéns a todas as equipes!

http://br.youtube.com/watch?v=WiOqB9aqhw0

Robótica no Município de Itajobi

Atividade desenvolvida com crianças do ensino publico no município de Itajobi no estado de São Paulo. O evento faz parte do projeto Nossa Cidade desenvolvido pela Nossa Caixa Nosso Banco.

http://br.youtube.com/watch?v=nB71FfITlGo

Equipe ATJV - Campus_Party

Robô construído por participantes da gincana da Campus Party 2008. Projeto da Equipe ATJV.

http://br.youtube.com/watch?v=CdF4t3C9Lj4

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Conectividade com ZeosDBO

Criando aplicações multi-banco com o ZeosDBO

Daniel Sobrinho Laporte
daniel.laporte@gmail.com
É editor técnico da revista ClubeDelphi. Graduado em Sistemas de informação e pós-graduado em administração de bancos de dados pela Universidade de Franca. É analista de sistemas e desenvolvedor Delphi desde 2005 desenvolvendo aplicações cliente/servidor e multicamadas em Delphi e Firebird.

Resumo do DevMan

Possibilitar a flexibilidade em suas aplicações é de extrema importância nos dias de hoje. E um dos principais quesitos que são levados em conta neste caso, trata da capacidade de ter aplicações que possam interagir com qualquer banco de dados sem grandes modificações no código.

A grande idéia deste artigo é exibir formas de tornar sua aplicação flexível, sem que haja a necessidade de adaptações. Daremos enfoque total a ferramenta Open Source, de grande qualidade e velocidade. Apresento-lhes o componente ZeosDBO.

Nesse artigo veremos
Criação de aplicações inteligentes, que possibilitem a integração entre múltiplos bancos de dados, adotando o componente OpenSource ZeosDBO;

Qual a finalidade
Exibir uma forma paralela ou diferenciada de se realizar conexões com diversos bancos de dados, visando assim possibilitar uma maior flexibilidade em futuras migrações.

Quais situações utilizam esse recurso?
Qualquer tipo de aplicação pode utilizar este componente juntamente com os recursos aqui mostrados. Aplicações multicamadas ou cliente-servidor, softwares que necessitam de grande compatibilidade entre os mais recentes bancos de dados disponíveis no mercado, aliados a alta performance e simplicidade.

Sempre que temos a missão de desenvolvermos um software, devemos fazer um processo minucioso de análise de todos os fatores que irão contribuir e influenciar na qualidade do nosso produto final. Na maioria dos casos, nós desenvolvedores nos preocupamos com fatores como: arquitetura na qual será aplicada, banco de dados ideal para o projeto, tecnologias que serão envolvidas, entre inúmeras outras. Destes fatores que citamos, podemos perceber que todos estão diretamente relacionados. Neste caso surgem algumas questões como: Qual componente de conexão devo utilizar? É possível integrar este projeto com outros bancos

de dados? O que devo fazer para manter minha aplicação multi-bancos sem a necessidade de ter que reescrevê-la? Muitos desenvolvedores poderiam citar a utilização do DBExpress, por exemplo, porém, alguns desenvolvedores preferem utilizar outros componentes, que possuem qualidade tão boa quanto esta citada.

Neste artigo, iremos abordar e responder a estas e muitas outras questões relativas à multi-conectividade, dicas de desenvolvimento para múltiplos bancos de dados, compatibilização da aplicação, preparação do ambiente. Tudo isso utilizando uma solução Open Source denominada de ZeosDBO.

O ZeosDBO (Zeos DataBase Objects), consiste em uma coleção de componentes para bancos de dados de grande qualidade, e atualmente é mantida por desenvolvedores do mundo todo. A grande vantagem deste componente (além de ser Open Source) se deve a alta quantidade de bancos de dados compatíveis com a biblioteca. Entre eles podemos citar: Firebird (versões 1.0, 1.5, 2.0 e 2.1), Interbase 5 e 6, MySQL, Oracle9i, PostgreSQL e SQLLite.

Exibiremos neste artigo, um passo a passo de como instalar o componente no Delphi, bem como sua utilização em conjunto com a tecnologia DataSnap. Estaremos substituindo o papel anteriormente desempenhado pelo DBExpress, por esta nova biblioteca de componentes, ilustrando algumas dicas importantes para manter esta conectividade entre os bancos de dados selecionados para o exemplo. Mãos a obra!

Preparando o ambiente

Para o nosso exemplo iremos utilizar dois bancos de dados distintos, bem como duas ferramentas de administração dos mesmos. Primeiramente iniciaremos pelo Firebird. Boa parte dos desenvolvedores que utilizam Delphi já conhecem este excelente banco de dados, porém, para quem ainda não conhece ou não utilizou, o Firebird é um banco de dados robusto, gratuito, de fácil instalação e manutenção. Utilizaremos em nosso artigo a versão 2.0.5 que está disponível para download no link http://sourceforge.net/project/downloading.php?groupname=firebird&filename=Firebird-2.0.5.13206_1_win32.exe&use_mirror=ufpr. Em seguida, faremos o download da ferramenta de administração. Para o meu caso, escolhi o IBExpert Personal que é gratuito e está disponível no link http://ibexpert.net/ibe/index.php?n=Main.PersonalEdition, mas existem outras ferramentas muito interessantes como o IBOConsole e o FlameRobin que são tão boas quanto esta que me refiro. Siga os passos indicados na página e faça a instalação da ferramenta. Passaremos agora para o MySQL. O MySQL assim como o Firebird, também é um excelente banco de dados que possui as mesmas características, sendo robusto, de fácil instalação e manutenção. A versão que utilizaremos será a 5.0 que está disponível no link http://dev.mysql.com/get/Downloads/MySQL-5.0/mysql-essential-5.0.83-win32.msi/from/pick, porém é necessário estar registrado. A ferramenta de administração que utilizarei para este banco de dados será o HeidiSQL, que também é gratuita e está disponível no link http://www.heidisql.com/download.php. Por último iremos baixar a versão do ZeosDBO que será mostrada no exemplo. Vale lembrar que já existe uma versão Alpha dos componentes para o Delphi 2009, porém como já é de nosso conhecimento, uma versão Alpha não deve ser utilizada em produção, portanto, iremos utilizar a última versão 6.6.5 Stable, na qual é um Release e será instalada no Delphi 2007. O link para o download do componente é: http://sourceforge.net/project/downloading.php?group_id=35994&filename=ZEOSDBO-6.6.5-stable.zip&a=67423289.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal DevMedia e assista a uma vídeo-aula do autor Luciano Pimenta, onde o mesmo mostra como utilizar a ferramenta IBExpert.

*http://www.devmedia.com.br/articles/viewcomp.asp?comp=3082&hl=*ibexpert**

Instalando o componente

Após ter feito o download do componente, crie uma pasta no diretório de sua preferência. Estarei me baseando em C:\Componentes\D2007\ZeosDBO para realizar sua extração. Descompacte os componentes na pasta anteriormente criada e em seguida abra seu RAD Studio 2007. Acesse o menu Tools>Options>Environment Options>Delphi Options>Library - Win32>... Na tela que é exibida, selecione o botão com três pontilhados para indicar o caminho das Unit's. Selecione a pasta src e adicione os caminhos correspondentes à component, core, dbc, parsesql e plain conforme ilustra a **Figura 1**.

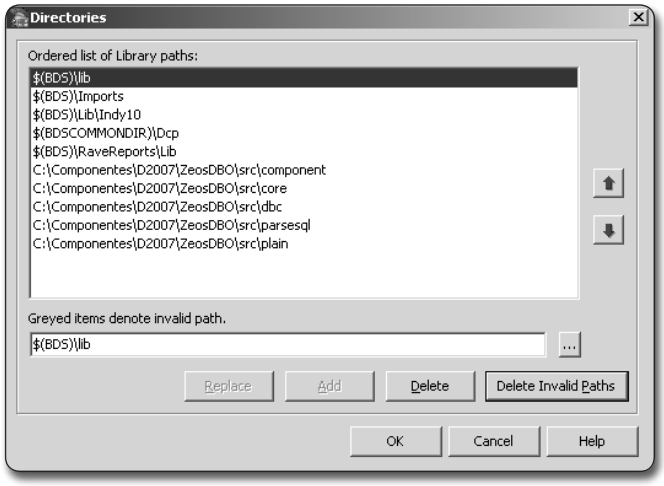


Figura 1. Diretórios das Units do componente

Após ter adicionado estas Unit's, iremos abrir o package de instalação do componente. Volte ao diretório inicial onde foram extraídos os arquivos e acesse a pasta packages e em seguida entre na pasta correspondente à versão do seu Delphi, que no nosso caso será delphi11 (RAD Studio 2007). Por último, abra o arquivo ZeosDbo.bdsproj. Compile todos os projetos, e ao concluir o processo de compilação, selecione o pacote ZComponentDesign110.bpl com o botão direito do mouse e escolha a opção Install. Se o processo for bem sucedido, você terá a exibição de uma mensagem semelhante à da **Figura 2**.

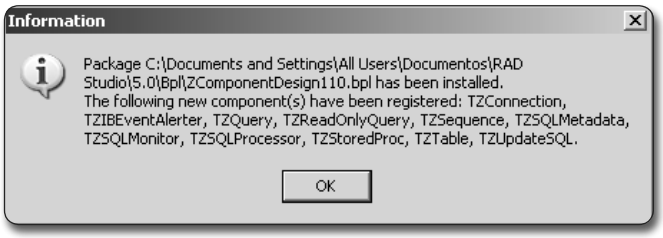


Figura 2. Imagem de confirmação da instalação

Após a instalação, você terá uma nova paleta de componentes conforme mostra a **Figura 3**.

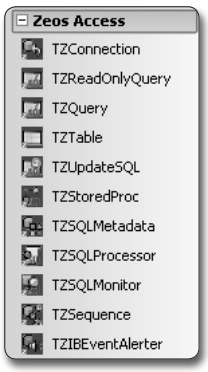


Figura 3. Paleta de componentes do ZeosDbo

Passaremos agora à etapa de criação dos dois bancos de dados.

Criando os banco de dados

Como o objetivo deste artigo não é criar uma aplicação, mas sim exibir a compatibilidade entre bancos de dados distintos, iremos então simular um simples cadastro de endereços que possua uma estrutura idêntica nos dois bancos. Além disto, modificaremos a maneira de realizar a obtenção dos valores auto-incrementais no banco para que assim, possamos ter nossa aplicação independente de objetos como Generator's, Sequence's ou mesmo do próprio auto-incremento do campo (no caso do MySQL). A **Figura 4** ilustra as duas tabelas que teremos em nossa aplicação. Observando, temos uma tabela simples e auto-explicativa (ENDERECO). A tabela CHAVE será responsável por controlar a organização das chaves primárias de nossa aplicação, baseando-se no campo de mesmo nome. O número sequencial será sempre incrementado e gravado nesta tabela. O script para a criação dos bancos de dados é exibido na **Listagem 1**, e devem ser executados e registrados nas ferramentas de administração.

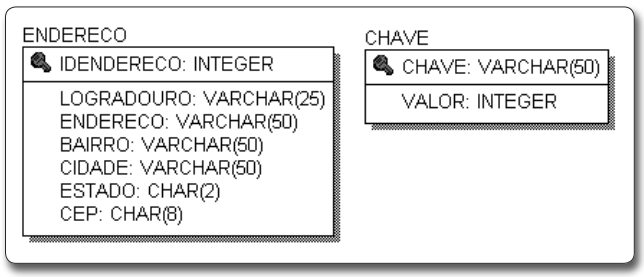


Figura 4. Tabelas do sistema

Listagem 1. Script de criação dos bancos de dados

```
[Firebird]
SET SQL DIALECT 3;
SET NAMES WIN1252;

SET CLIENTLIB '<DIRETÓRIO DA INSTALAÇÃO DO FIREBIRD>\BIN\
fbclient.dll';

CREATE DATABASE '<DIRETÓRIO DO BANCO DE DADOS>\ZEOS.FDB'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 4096

DEFAULT CHARACTER SET WIN1252;

CREATE TABLE ENDERECO (
    IDENDERECO  INTEGER NOT NULL,
    LOGRADOURO  VARCHAR(25) NOT NULL,
    ENDERECO    VARCHAR(50) NOT NULL,
    BAIRRO      VARCHAR(50) NOT NULL,
    CIDADE      VARCHAR(50) NOT NULL,
    ESTADO      CHAR(2) NOT NULL,
    CEP         CHAR(8)
);

ALTER TABLE ENDERECO ADD CONSTRAINT
PK_IDENDERECO PRIMARY KEY (IDENDERECO)
USING INDEX IDX_IDENDERECO;

CREATE TABLE CHAVE (
    CHAVE  VARCHAR(50) NOT NULL,
    VALOR  INTEGER NOT NULL
);

ALTER TABLE CHAVE ADD CONSTRAINT
PK_CHAVE PRIMARY KEY (CHAVE)
USING INDEX IDX_CHAVE;

[MySQL]
CREATE DATABASE `ZEOS` CHARACTER SET LATIN1;

CREATE TABLE ENDERECO(
    IDENDERECO  INT(11) NOT NULL,
    LOGRADOURO  VARCHAR(25) NOT NULL,
    ENDERECO    VARCHAR(50) NOT NULL,
    BAIRRO      VARCHAR(50) NOT NULL,
    CIDADE      VARCHAR(50) NOT NULL,
    ESTADO      CHAR(2) NOT NULL,
    CEP         CHAR(8),
    PRIMARY KEY (IDENDERECO)
)ENGINE=INNODB DEFAULT CHARSET LATIN1;

CREATE TABLE CHAVE(
  CHAVE VARCHAR(50) NOT NULL,
  VALOR INT(11) NOT NULL,
  PRIMARY KEY(CHAVE)
)ENGINE=INNODB DEFAULT CHARSET LATIN1;
```

Desenvolvendo o exemplo

Abra seu RAD Studio 2007 para iniciarmos a nossa aplicação. Vá no menu File>New>VCL Forms Application – Delphi for Win32, no novo Form (FrmPrincipal), altere sua propriedade Caption para Principal, Position para poDesktopCenter e WindowState para wsMaximized. Adicione um componente MainMenu (Menu), por último adicione um item de menu com a descrição de Endereços. Salve seu projeto nomeando-o como AppZeos.dproj e a Unit como untPrincipal.pas. Novamente acesse o menu File>New>Other>Delphi Files e do lado direito selecione um novo Data Module (DMPrincipal). Daremos início à inserção de componentes. Adicione ao novo Data Module um ZConnection (zConnection), três ZQuery (zqryEndereco, zqryConsulta, zqryChave), três DataSetProvider (dspEndereco, dspConsulta, dspChave) e um ClientDataSet (cdsEndereco, cdsConsulta, cdsChave). Primeiramente crie no zConnection uma conexão com o Firebird 2 apenas para configurações futuras. Ligue os ZQuery's ao zConnection através da propriedade Connection, em seguida aponte a propriedade DataSet dos DataSetProvider's a suas respectivas ZQuery's e por último

ligue as propriedades ProviderName dos componentes ClientDataSet aos respectivos DataSetProvider's para que assim tenhamos realizado as ligações dos componentes. Terminando esta etapa, adicione às ZQuery's através da propriedade SQL as instruções correspondentes, e detalhadas na **Listagem 2**.

Listagem 2. Instruções SQL para os componentes ZQuery

```
[zqryEndereco]
SELECT
  ENDERECO.IDENDERECO,
  ENDERECO.LOGRADOURO,
  ENDERECO.ENDERECO,
  ENDERECO.BAIRRO,
  ENDERECO.CIDADE,
  ENDERECO.ESTADO,
  ENDERECO.CEP
FROM
  ENDERECO
WHERE
  ENDERECO.IDENDERECO =:IDENDERECO

[zqryChave]
SELECT
  CHAVE.CHAVE,
  CHAVE.VALOR
FROM
  CHAVE
WHERE
  CHAVE.CHAVE = :CHAVE
```

Nota

A conexão do componente com o banco de dados foi realizada apenas para efeito de configurações, no decorrer do artigo iremos desenvolver o modelo de conexão para os demais bancos de dados, que poderão ser reutilizados por outros modelos de base de dados que não foram abordadas neste artigo.

Após ter adicionado as instruções, selecione o zqryEndereco e clique com o botão direito do mouse sobre ele, abrindo o Fields Editor. Novamente clique com o botão direito, agora no FieldsEditor e selecione Add Al lFields e configure os ProviderFlags referentes a cada um dos campos, lembrando que o campo chave (IDENDERECO) deve estar habilitado as opções [pfInkey, pfInUpdate, pfInWhere] e os demais campos só devem possuir [pfInUpdate]. Repita este mesmo processo para o zqryChave. Passando para os DataSetProvider's, configure no dspChave e no dspEndereco a propriedade UpdateMode para upWhereKeyOnly, e em Options marque [poIncFieldProps, poPropagateChanges] e para o dspConsulta altere apenas em Options [poAllowCommandText], finalizando assim as configurações destes componentes. Por último adicione os campos no Fields Editor dos ClientDataSet's assim como feito anteriormente nos ZQuery's. Assim finalizamos as configurações iniciais, passaremos agora para a parte de codificação que deve ser realizada neste Data Module.

Iniciando pelas configurações mais simples, coloque o seguinte código no evento OnDestroy do nosso Data Module:

```
zConnection.Connected := False;
```

Selecione os eventos AfterDelete e AfterPost do cdsEndereco e digite o código que confirmará a gravação no banco de dados, conforme exibido a seguir:

```
TClientDataSet(DataSet).ApplyUpdates(0);
```

Passaremos agora para duas funções nas quais fundamentam as maiores observações do nosso projeto. Primeiramente criaremos um método na qual será responsável pela configuração e validação da conexão que será criada pelo usuário. Crie no escopo public de seu Data Module o cabeçalho da rotina exibida na **Listagem 3**.

Listagem 3. Código da rotina TestConnection

```
01 function TDMPrincipal.TestConnection(DataSet : TClientDataSet) : Boolean;
02 begin
03   Result := False;
04   if Assigned(DataSet) then
05   begin
06     try
07       zConnection.Connected := False;
08       zConnection.Protocol  := DataSet.FieldByName('Protocol').AsString;
09       zConnection.HostName  := DataSet.FieldByName('Host').AsString;
10       zConnection.Port      := DataSet.FieldByName('Port').AsInteger;
11       zConnection.Database  := DataSet.FieldByName('Database').AsString;
12       zConnection.Password  := DataSet.FieldByName('Pass').AsString;
13       zConnection.User      := DataSet.FieldByName('User').AsString;
14       zConnection.Connected := True;
15       Result := True;
16     except
17       on e: exception do
18       begin
19         Result := False;
20         ShowMessage('Erro de conexão : ' + e.Message);
21       end;
22     end;
23   end;
24 end;
```

Reparando no código da nossa **Listagem 3**, podemos notar que é recebido um ClientDataSet através do parâmetro DataSet. Na **Linha 04** é realizado um teste de prevenção, que garante se o objeto passado foi criado, ou não está nulo. Entre as **Linhas 07** a **13**, temos a desconexão e passagem dos parâmetros fundamentais para a configuração do componente zConnecion, desta forma temos muita flexibilidade em alterações e criação de novas conexões. Logo na **Linha 14** é tentada a conexão, caso não ocorra nenhum problema esta nossa função retornará o valor boleano True, caso contrário o retorno será False e uma mensagem com o erro ocorrido será exibida (**Linhas 16** a **22**).

Nota

Em ambientes multicamadas, ou mesmo em aplicações multi-banco esse tipo de configuração é comum, sendo realizada por meios de arquivos .ini ou .xml.

Feito isto, teremos realizado o grande detalhe do nosso artigo, porém, ainda nos resta a criação da tela responsável por esta configuração, na qual será detalhada mais adiante.

O próximo código é um exemplo bem interessante e alternativo à criação de objetos ou meios de obtenção do autoincremento pela base dados. Desta forma, podemos dispensar a utilização de campos auto-incrementais, Generator's, Sequence's, Trigger's ou outras maneiras para realizar esta tarefa. A grande vantagem é a centralização desta tarefa por parte da aplicação, tornando-a independente da base de dados escolhida. Novamente declare o método no escopo public do DMPrincipal. Veja o código detalhado na **Listagem 4**.

Listagem 4. Código da rotina GetKeyValue

```
01 function TDMPrincipal.GetKeyValue(KeyName: String): Integer;
02 begin
03   Result := -1;
04   if Trim(KeyName) <> '' then
05   begin
06     try
07       repeat
08         cdsChave.Close;
09         cdsChave.FetchParams;
10         cdsChave.Params[0].AsString := UpperCase(KeyName);
11         cdsChave.Open;
12         if cdsChaveVALOR.Value > 0 then
13         begin
14           cdsChave.Edit;
15           cdsChaveVALOR.Value := cdsChaveValor.Value + 1;
16           cdsChaveCHAVE.AsString := UpperCase(KeyName);
17         end
18         else
19         begin
20           cdsChave.Append;
21           cdsChaveVALOR.Value := 1;
22           cdsChaveCHAVE.AsString := UpperCase(KeyName);
23         end;
24         cdsChave.Post;
25         Result := cdsChaveVALOR.Value;
26       until cdsChave.ApplyUpdates(0) = 0;
27     except
28       Result := -1;
29     end;
30   end;
31 end;
```

O parâmetro de entrada desta função (KeyName), como o próprio nome sugere, é o nome da chave primária. Na **Linha 04** é realizada a validação que verifica se a chave digitada não está em branco, caso a condição seja satisfeita teremos um bloco de repetição contínua. Entre as **Linhas 08** e **11** é realizada a passagem de parâmetros para o cdsChave, baseando-se no campo recebido por parâmetro. Desta forma o ClientDataSet realiza uma consulta considerando este mesmo parâmetro. Se o resultado do campo VALOR (referente ao ClientDataSet) for maior que zero, então o registro é editado e tem seu conteúdo somado a um (**Linhas 12** a **17**), caso contrário é inserido um novo registro como o número de chave inicial (**Linhas 18** a **23**). Se houver violação de chave primária, ou algum outro problema, a nossa instrução será repetida novamente, tendo como base o valor do ApplyUpdates que deve ser igual a zero (**Linha 26**). De qualquer forma, caso não seja possível obter o valor, nossa rotina retornará o valor -1.

Nota do DevMan

Em ambientes multicamadas, é comum o tipo de instrução semelhante ao da rotina GetKeyValue. Isso se deve a própria sugestão do modelo, onde tem por objetivo possuir total independência, tornando a aplicação escalável e com grande portabilidade. Utilizando técnicas como esta, é possível manter uma mesma aplicação compatível qualquer banco de dados relacional atual, pois aborda o conceito de tabelas, na qual é comum a todos.

Nota do DevMan

Uma questão que deve ser levada em conta quando se objetiva trabalhar com múltiplos bancos de dados é a criação dos campos no Fields Editor. Muitos desenvolvedores possuem receios quanto a esta prática, porém, é perfeitamente possível que se tenha campos pré-definidos em um DataSet, contanto que haja alguns cuidados. Observando este artigo, podemos notar que os campos foram criados (em ambas as bases de dados) com tipos compatíveis entre si, garantindo assim a prevenção de problemas na estrutura física ou mesmo no tipo de dado definido no Fields Editor.

Nota do DevMan

É comum entre desenvolvedores e/ou DBA's utilizar recursos de Sequence (Oracle, Firebird), Generator (Firebird) ou campos AUTO_INCREMENT (MySQL). Esta prática não deve ser dispensada, porém, antes de aplicá-la devemos levar em conta vários fatores como diversificação de bancos de dados ou compatibilidade, caso contrário, será necessário reescrever estes objetos, e futuramente dificultando a manutenção em múltiplos bancos de dados diferentes.

Para finalizarmos o Data Module, iremos definir a instrução de auto-incremento baseada na rotina em que criamos anteriormente. Implemente o código a seguir no evento AfterInsert do cdsEndereco:

```
cdsEndereco.FieldByName('IDENDERECO').Value :=
GetKeyValue('IDENDERECO');
```

Salve seu projeto nomeando a nova Unit como untDMPrincipal.pas. Iremos agora criar a tela de configuração da conexão, na qual será responsável pela alimentação da rotina TestConnection. Para isso crie um novo formulário através do menu File>New>Form - Delphi for Win32. Inicie nomeando-o para frmBanco e altere seu Caption para Selecione o Banco de Dados. Em seguida, insira um ClientDataSet (cdsConexao), um DataSource (dsrConexao), uma DBGrid, um DBNavigator e dois BitBtn's (btnConectar, btnCancelar). Dê um duplo clique sobre o cdsConexao e no FieldsEditor e crie seis novos campos (Host, DataBase, User, Pass, Port e Protocol) sendo apenas o campo Port do tipo Integer e os demais String com tamanho de 100 caracteres. Ligue a DBGrid e o DBNavigator ao DataSource através da propriedade de mesmo nome, e em seguida atribua o cdsConexao à propriedade DataSet do DataSource. Ao final da configuração veja como ficou nossa interface conforme ilustra a **Figura 5**.

Figura 5. Interface da tela de configuração de conexões

Já no desenvolvimento, adicione à cláusula Uses do novo formulário a untDMPrincipal e crie uma variável (stPath) do tipo String no escopo private. No evento OnCreate do mesmo, faça a seguinte codificação:

```
stPath := ExtractFilePath(Application.ExeName) +
'Connection.xml';
if FileExists(stPath) then
  cdsConexao.LoadFromFile(stPath)
else
  cdsConexao.CreateDataSet;
```

No evento OnCreate é verificada a existência do arquivo denominado como Connection.xml, localizado no diretório da aplicação, caso exista, este será automaticamente carregado, caso contrário é realizada a preparação para inserção no cdsConexao para que futuramente este arquivo seja criado. Observe que a variável stPath obteve sua inicialização como o diretório da aplicação concatenado ao nome do arquivo que desejamos criar. Para definirmos a instrução que criará o arquivo, faça a seguinte implementação no evento AfterPost do cdsConexao:

```
cdsConexao.SaveToFile(stPath, dfXML);
```

Por último, adicione ao evento OnClick do btnConectar o código a seguir, responsável por validar a nossa conexão criada através da função anteriormente definida no DMPrincipal (TestConnection).

```
if (cdsConexao.Active) and (cdsConexao.RecordCount
> 0) then
begin
  if DMPrincipal.TestConnection(cdsConexao) then
    Self.ModalResult := mrOk
  else
    Self.ModalResult := mrCancel;
end;
```

Repare que antes de testar a conexão, verificamos se o cdsConexao está ativo, e se existem registros cadastrados. O resultado da rotina TestConnection determinará se nossa aplicação continuará seu ciclo normalmente, ou finalizada após o resultado False. Com isso finalizamos a configuração deste formulário. Prosseguiremos agora criando o cadastro e a consulta da nossa aplicação de exemplo.

Crie um novo Form (frmPesquisa) e modifique suas propriedades Position para poMainFormCenter e Caption para Consulta. Em seguida adicione ao formulário um ComboBox (cmbParametro), um Edit (edtPesquisa), um BitBtn (btnPesquisar), um DataSource (dsrPesquisa) e um DBGrid. Modifique suas propriedade Align para alBottom, DataSource para dsrPesquisa e ReadOnly para True. Limpe as propriedades Text do edtPesquisa e do cmbParametro para concluir a adição de componentes. Veja na **Figura 6** o resultado final de nossa interface.

O objetivo da nossa tela de pesquisa é ser utilizada em qualquer eventual cadastro, portanto criaremos uma forma que nossa tela se encarregará de montar a instrução SQL conforme alguns parâmetros que definirmos. Primeiramente declare uma variável do tipo StringList (FStringList) no escopo private do nosso Form. Declare na sessão uses a Unit DBClient e em seguida crie duas propriedades na sessão public conforme mostrado a seguir:

```
property DataSetPesquisa : TClientDataSet;
property SQL : String;
```

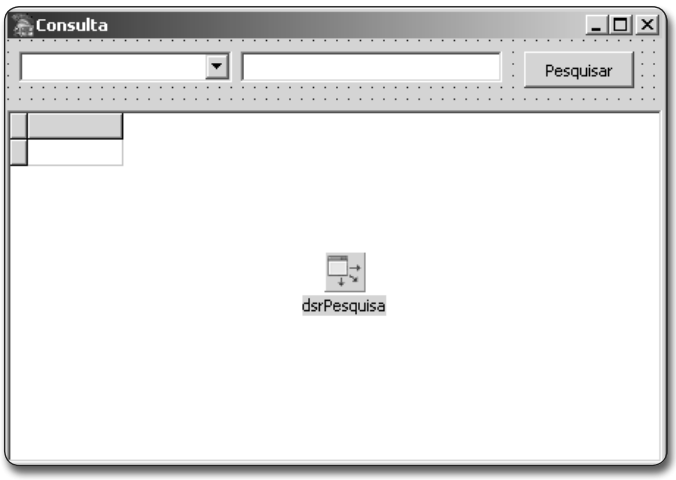


Figura 6. Interface da tela de consultas

Após sua declaração, pressione simultaneamente as teclas Ctrl + Shift + C para que os métodos sejam criados automaticamente. Altere o código do método SetDataSetPesquisa, adicionando as instruções a seguir:

```
FDataSetPesquisa := Value;
if Assigned(FDataSetPesquisa) then
  dsrPesquisa.DataSet := FDataSetPesquisa;
```

Faça agora as alterações no método SetSQL conforme exibe a próxima listagem:

```
FSQL := Value;
if Trim(FSQL) = '' then
  ShowMessage('Não foi definida a pesquisa !');
```

No evento OnCreate do frmPesquisa, crie o objeto StringList:

```
FStringList := TStringList.Create;
```

No evento OnShow, faça como que o CmbParametro seja iniciado na primeira posição:

```
cmbParametro.ItemIndex := 0;
```

O último evento do formulário será o OnDestroy, na qual realizará a destruição do StringList e o fechamento do DataSet de pesquisa:

```
FreeAndNil(FStringList);
FDataSetPesquisa.Close;
```

Criaremos agora o método responsável pela adição dos campos para a montagem da instrução SQL bem como seus descritivos que serão adicionados ao ComboBox. Declare seu cabeçalho na sessão public do formulário e implemente-o conforme a **Listagem 5**.

Listagem 5. Método AddFieldsToCombo

```
01 procedure TfrmPesquisa.AddFieldsToCombo(Campo: String);
02 var
03   i : Integer;
04   stField,
05   stLabel : String;
06   boControle : Boolean;
07 begin
08   stField := '';
09   stLabel := '';
10   boControle := False;
11   for i := 1 to Length(Campo) do
12   begin
13     if Campo[i] = ';' then
14       boControle := True;
15     if not boControle then
16       stLabel := stLabel + Campo[i]
17     else if (boControle) and (Campo[i] <> ';') then
18       stField := stField + Campo[i];
19   end;
20   cmbParametro.Items.Add(stLabel);
21   FStringList.Add(stField);
22 end;
```

Perceba na **Listagem 5**, que a rotina possui um parâmetro de entrada denominado Campo. Mais adiante, veremos a forma em que devemos manipular este parâmetro para a correta montagem da instrução. Na **Linha 11** iniciamos uma contagem de caracteres na String, em seguida verificamos o nosso delimitador (;) que será responsável por separar a instrução do nome do campo e a instrução do Caption (**Linhas 13** a **18**). Desta forma o conteúdo descritivo será adicionado à variável stLabel, e o conteúdo do campo será atribuído ao stField. Por último o ComboBox será alimentado com o valor descritivo e a StringList será alimentada com o conteúdo do nome do campo (**Linhas 20** e **21**).

O retorno da nossa pesquisa será realizado no momento em que o usuário selecionar o registro através de um duplo clique na DBGrid, portanto, adicione o código que se segue, ao evento OnDblClick da mesma:

```
if (FDataSetPesquisa.Active) and (not
FDataSetPesquisa.IsEmpty) then
  ModalResult := mrOk
else
  ModalResult := mrCancel;
```

Para concluirmos esta tela faremos a implementação do evento OnClick no botão btnPesquisar, conforme mostra a **Listagem 6**.

Listagem 6. Método de consulta do botão Pesquisar

```
01 procedure TfrmPesquisa.btnPesquisarClick(Sender: TObject);
02 var
03   stParameter, stOperador : String;
04 begin
05   if Trim(edtPesquisa.Text) <> '' then
06   begin
07     stOperador  := '';
08     stParameter := '';
09     FDataSetPesquisa.Close;
10     FDataSetPesquisa.CommandText := FSQL;
11     case cmbParametro.ItemIndex of
12       0 :
13       begin
14         stOperador := ' = ';
15         stParameter := edtPesquisa.Text;
16       end
17       else
18       begin
19         stOperador := ' LIKE ';
20         stParameter := QuotedStr(edtPesquisa.Text) + ' || ' +
            QuotedStr('%');
21       end;
22     end;
23     FDataSetPesquisa.CommandText :=
       FSQL + ' where ' +
       FStringList.Strings[cmbParametro.ItemIndex] +
       stOperador + stParameter;
24     FDataSetPesquisa.Open;
25   end;
26 end;
```

Observando a **Listagem 6**, temos primeiramente a verificação do conteúdo digitado no Edit (**Linha 05**). A propriedade SQL do formulário se encarregará de receber a instrução principal neste caso, conforme a **Linha 10**. Em nosso exemplo o primeiro item do ComboBox fará referência ao código, neste caso o operador de instrução da condição Where é o símbolo de igualdade (=), para os outros casos será considerado o operador LIKE acrescido de um caractere (%) (**Linhas 11** a **21**). Finalmente, na **Linha 23** temos a montagem da instrução SQL e por último a abertura da pesquisa como mostra a **Linha 24**. Salve sua Unit com o nome de untPesquisa.pas para darmos início ao último formulário de nossa aplicação.

Crie um novo Form (frmCadEndereco) e altere suas propriedades Caption para Cadastro de Endereços e Position para poMainFormCenter. Adicione ao Uses a Unit untPesquisa e em seguida insira um DataSource (dsrCadastro), sete Label's (Código, Logradouro, Endereço, Bairro, Cidade, Estado, CEP), seis DBEdit's, um DBText e por último um BitBtn (btnPesquisar). Ligue o dsrCadastro ao DBNavigator, DBText e DBEdit's através da propriedade DataSource. Em seguida faça a linkagem dos campos através da propriedade DataField.

Nota

Nota: Como o vínculo do cdsEnderecos ao dsrCadastro será feito em RunTime, será necessário digitar informar os campos manualmente nas propriedades DataField tanto para o componente DBText quanto para os DBEdit's.

Vamos agora concluir nossa aplicação adequando-a aos códigos restantes. Primeiramente na sessão public, declare duas propriedades:

```
{ Adicione DBClient à Uses}
 property DataSetCadastro : TClientDataSet;
 property DataSetPesquisa : TClientDataSet;
```

Novamente pressione Ctrl + Shift + C, e no método SetDataSetCadastro faça a seguinte alteração:

```
FDataSetCadastro := Value;
if Assigned(FDataSetCadastro) then
begin
  dsrCadastro.DataSet := FDataSetCadastro;
  FDataSetCadastro.Open;
end;
```

Finalmente implementaremos o evento que chama a pesquisa no evento OnClick do botão btnPesquisar como exibido na **Listagem 7**.

Vamos ao código da **Listagem 7**. O primeiro detalhe interessante do código ocorre após a **Linha 07**, onde fizemos a criação da tela e pesquisa em Runtime. Nas **Linhas 11** e **12**, o método AddFieldsToCombo é chamado, e como explicado anteriormente temos na primeira instrução antes do delimitador (;) o nome que será exibido no ComboBox para consulta e em seguida o nome da tabela junto ao campo que pertence.

Listagem 7. Método de consulta do botão Pesquisar no cadastro de endereços

```
01 procedure TfrmCadEndereco.btnPesquisarClick(Sender: TObject);
02 var
03   i, j : Integer;
04 begin
05   if not(DataSetCadastro.State in [dsEdit, dsInsert]) then
06   begin
07     frmPesquisa := TfrmPesquisa.Create(Self);
08     with frmPesquisa do
09     try
10       DataSetPesquisa := Self.DataSetPesquisa;
11       AddFieldsToCombo('Código;ENDERECO.IDENDERECO');
12       AddFieldsToCombo('Endereço;ENDERECO.ENDERECO');
13       SQL := ' SELECT ENDERECO.* FROM ENDERECO ';
14       if ShowModal = mrOK then
15       begin
16         DataSetCadastro.Close;
17         DataSetCadastro.FetchParams;
18         for i := 0 to DataSetCadastro.Params.Count - 1 do
19         begin
20           for j := 0 to DataSetPesquisa.Fields.Count - 1 do
21           begin
22             if UpperCase(DataSetCadastro.Params[i].Name) =
               UpperCase(DataSetPesquisa.Fields[j].FieldName) then
23               begin
24               DataSetCadastro.Params[i].Assign
                   (DataSetPesquisa.Fields[j]);
25               break;
26             end;
27           end;
28         end;
29         DataSetCadastro.Open;
30       end;
31     finally
32       FreeAndNil(frmPesquisa);
33     end;
34   end;
35 end;
```

Na próxima linha, (**Linha 13**) a propriedade SQL recebe a instrução padrão que será utilizada para que a tela de pesquisa se encarregue de montar a parametrização. Conforme o retorno de mrOk obtido da tela, o DataSet é fechado e tem seu parâmetro carregado (**Linhas 16** e **17**) . Entre as **Linhas 18** e **28** temos uma instrução de loop onde verifica-se se o nome do campo contido na pesquisa (tela de consulta) é idêntico ao nome esperado pelo parâmetro. Caso este seja encontrado, seu valor automaticamente é copiado para dentro do parâmetro e a instrução é parada através do break (**Linha 25**). Na **Linha 29** teremos a abertura do DataSetCadastro e finalmente a destruição e liberação do objeto de pesquisa da memória (**Linha 32**). Salve seu projeto nomeando a nova Unit como untCadEndereco.pas.

Volte ao frmPrincipal para fazermos as chamadas aos formulários. Instancie o evento OnCreate conforme a listagem que se segue:

```
procedure TfrmPrincipal.FormCreate(Sender:
TObject);
begin
  frmBanco := TfrmBanco.Create(Self);
  with frmBanco do
  try
    if ShowModal <> mrOk then
      Application.Terminate;
  finally
    FreeAndNil(frmBanco);
  end;
end;
```

No item que criamos no componente MainMenu adicione a chamada ao nosso cadastro de endereços:

```
procedure TfrmPrincipal.Endereos1Click(Sender:
TObject);
begin
  frmCadEndereco := TfrmCadEndereco.Create(Self);
  with frmCadEndereco do
  try
    DataSetCadastro := DMPrincipal.cdsEndereco;
    DataSetPesquisa := DMPrincipal.cdsConsulta;
    ShowModal;
  finally
    FreeAndNil(frmCadEndereco);
  end;
end;
```

Para finalizar, configure sua aplicação no menu Project>Options>Forms como exibe a **Figura 7**.

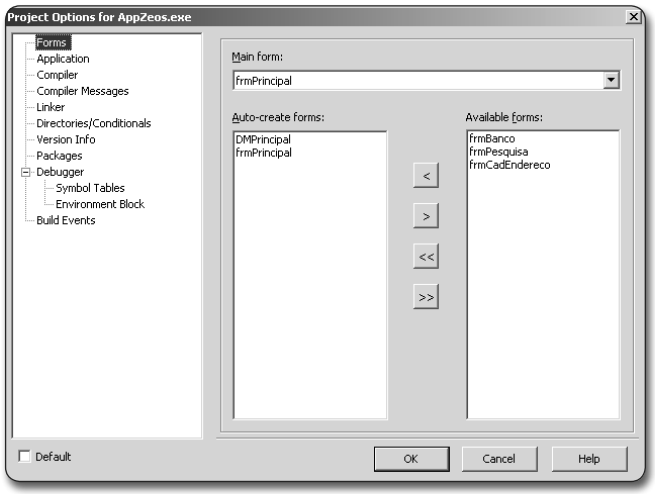


Figura 7. Configuração do projeto

Conclusão

Como pudemos perceber o ZeosDBO é uma ótima alternativa para aplicações multi-bancos de dados. Se você não é adepto da tecnologia DBExpress e apóia as tecnologias OpenSource, sem dúvida alguma este componente é uma boa pedida, uma vez que possui requisitos fundamentais para quem procura qualidade no desenvolvimento de software. Vale lembrar que neste artigo utilizamos a tecnologia DataSnap em conjunto com a biblioteca, porém é perfeitamente possível utilizar apenas a biblioteca para desenvolvimento. Mais uma vez espero que tenham gostado do artigo. Um grande abraço e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Delphi e PostGreSQL

Como desenvolver aplicações utilizando o banco de dados free PostGreSQL?

Adriano Santos
falecom@adrianosantos.pro.br
É desenvolvedor Delphi desde 1998, professor e programador PHP, bacharel em Comunicação Social e especialista em aplicações Win32. É Editor Chefe das revistas ClubeDelphi e WebMobile. Gerente de TI na SoftPark Informática e membro fundador do DUG-SP (www.dug-sp.com). Mantenedor do blog Delphi to Delphi (www.delphitodelphi.blogspot.com) com dicas, informações e tudo sobre o desenvolvimento Delphi.

Resumo do DevMan

Existem n bancos de dados free no mercado, entre eles Firebird, MySQL e PostGreeSQL. Digamos que esses três bancos de dados concorrem fortemente no mercado abocanhando boa parte da preferência de desenvolvedores em todo mundo.

Nesse artigo abordarei o uso do PostGreSQL em aplicações Delphi e servidores Windows. Não haverá grandes detalhamentos, apenas uma pequena introdução ao BD e ao driver de acesso no RAD Studio 2009.

Nesse artigo veremos
Esse artigo trata de uma apresentação rápida do banco de dados PostGreSQL.

Qual a finalidade
O PostGreSQL é um banco de dados gratuito disponível na internet para uso em qualquer plataforma e linguagem de programação.

Quais situações utilizam esse recurso?
De fato podemos usar os conhecimentos adquiridos aqui em qualquer situação, desde o desenvolvimento de pequenos sistemas até grandes softwares.

Já há algum tempo venho recebendo e-mails de colegas solicitando um artigo para demonstrar o uso do banco de dados PostGreSQL com Delphi e confesso que não sabia muito bem como fazê-lo. Recentemente participei de um projeto que necessitava fazer essa conexão e desenvolver o software pensando nesse BD. Foi ai que resolvi estudá-lo e descobri o porquê de tantas mensagens. Para minha surpresa o SGBD, que iniciou sua trajetória de sucesso com muita timidez, se mostrou bastante estável e rápido. Ainda não tenho informações concretas sobre a sua performance em produção, mas posso afirmar que até agora não tive grandes problemas com ele.

Tenho percebido uma boa aceitação do banco em órgãos públicos, por incrível que pareça, e há até aqueles mais ousados que comparam o PostGreSQL com Oracle. Mas isso não vem ao caso. O fato é que esse elefantinho simpático tem ganho adeptos em todas as áreas e não podemos deixar de citá-lo.

A ideia desse artigo não é esmiuçar o banco, mas mostrar como fazer o acesso via Delphi de maneira simples e objetiva

para que todos tenham uma base de como proceder. Vamos ver como fazer o download e instalação em um Windows Server, criação do banco teste e acesso a suas tabelas via aplicação Delphi.

Um pouco de história

O PostGreSQL é mais antigo do que parece. Na verdade ele é fruto do projeto Ingres desenvolvido pela Universidade de Berkeley da Califórnia. Michael Stonebraker era o líder do projeto e um dos pioneiros em bancos de dados relacionais. Ele ausentou-se da universidade em 1982 para comercializar o Ingres, porém mais tarde voltou a universidade e após isso, em 1985, passou a desenvolver o projeto pós-Ingres. Seu objetivo foi resolver alguns problemas em tipos de dados no modelo relacional do Ingres. A incapacidade em compreender tipos, também chamados de objetos, era o principal problema que ser enfrentado.

O resultado de todo o estudo e correções das incapacidades do pós-Ingres passou a chamar-se de Postgres. Introduziu-se o mínimo de funcionalidades possível para então suportar tipos. O grande Q na época era que as funcionalidades deveriam incluir habilidades suficientes para definir-se tipos e também a possibilidade de descrever relações que, segundo algumas informações obtidas por outras fontes, até o momento eram amplamente utilizadas, mas mantidas por usuários. Em resumo, o Postgres entendia as relações e obtinha informações de tabelas relacionadas utilizando certas regras.

No início de 1986, vários documentos descrevendo a base do sistema foram divulgados e em 1988 já existia um protótipo do banco totalmente funcional. Finalmente a versão 1 foi liberada em 1989 e na sequência a versão 2, onde o sistema de regras foi totalmente reescrito. A versão 3 foi liberada em 1991 novamente com seu sistema de regras reescrito e dessa vez havia até suporte a múltimos gerenciadores de armazenamento. Um motor avançado de consultas também foi introduzido. Em 1993 o Postegres começava a fazer barulho e ganhar popularidade. Já havia muita demanda e solicitação de suporte. Com o lançamento da versão 4 a universidade abandonou o projeto.

Porém, como o código era aberto sob a licença BSD, o desenvolvimento acabou sendo continuado por entusiastas. Os estudantes Andrew Yu e Jolly Chen, também da universidade Berkeley, incluíram um interpretador SQL substituindo a linguagem QUEL que havia sido construída para o Ingress. Surgiu então o Postgres95. Seu código então passou a ser divulgado na internet e nesse momento virou Open Source.

A primeira versão depois do abandono da Universidade de Berkeley foi lançada por Marc Fournier, Bruce Momjian e Vadim B. Mikheev que por sua vez iniciaram uma da tarefas mais difícieis: estabilizar o código do banco de dados. Isso tudo aconteceu em 1996 e ainda nesse mesmo ano o projeto foi renomeado para PostGreSQL justamente pelo fato de já incorporar o interpretador SQL. A primeira versão foi liberada em janeiro de 1997 já com o número 6.0. Deste ano em diante o projeto tem sido mantido por desenvolvedores voluntários de todo o mundo.

Mais informações sobre o PostGreSQL podem ser obtidas também no site www.wikipedia.org ou em www.postgresql.org.br, comunidade brasileira.

Adquirindo e instalando o PostGreSQL

Agora que já temos uma boa noção desse banco de dados, é hora de adquirimos e o instalarmos para então desenvolvermos as primeiras aplicações com ele. O PostGreSQL é um banco de dados gratuito como pudemos comprovar e pode ser obtido no site www.postgresql.org ou ainda na comunidade brasileira www.postgresql.org.br.

Nesses links podemos encontrar todas as informações e versões para instalação. Acesse qualquer um dos links citados e logo em seguida clique em Downloads na parte superior do portal. Na sequência observe que existem versões do PostGreSQL para FreeBSD, Linux, MacOS, Solaris e Windows. Clique em Windows e aguarde o carregamento da página de download. Nós temos basicamente dois tipos de instaladores. Usaremos o primeiro deles que é mais simples e não requer conhecimento avançado, portanto clique em Download the one click installer. Você será redirecionado para a página principal de downloads, onde podemos adquirir a versão 8.3.7.1 que até o fechamento dessa edição era a versão mais atual. E a versão 8.4 de testes. Baixe a primeira delas.

Efetuado o download, execute o arquivo postgresql-8.3.7-1-windows.exe e aguarde o início da instalação. Não é difícil instalar o banco, mas é bom prestar bastante atenção nas telas do Wizard.

Na primeira tela (**Figura 1**) apenas clique em Next para que a tela de boas vindas seja pulada. Nas **Figuras 2** e **3**, necessitamos fazer a indicação do diretório de instalação e, respectivamente, o diretório que conterá os bancos de dados propriamente ditos. Para essa primeira apresentação, apenas mantenha-os default.

A **Figura 4** como podemos ver é a configuração de senha de acesso. O usuário administrador é sempre postgres com a senha que você configurar nesse momento. Digite então sua senha de preferência e clique em Next para continuar.

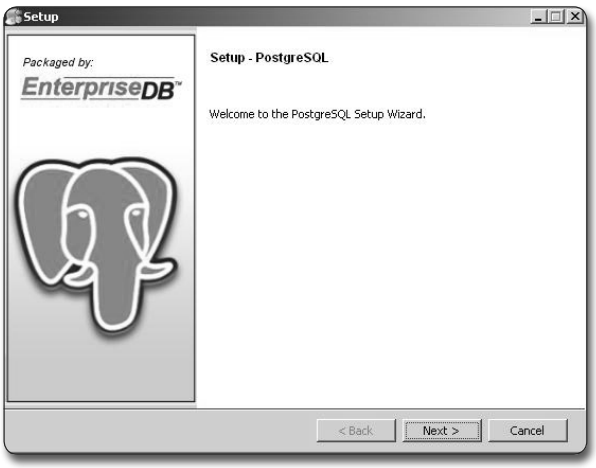


Figura 1. Tela de boas vindas

Logo em seguida, o PostGreSQL sugere a porta onde o servidor estará disponível. A porta padrão é a 5432 como podemos ver na **Figura 5**. É nesse momento que podemos fazer a mudança da porta utilizada. Caso não queria modificar agora, também é possível configurar mais tarde no arquivo de configurações do BD. Logo em seguida devemos configurar a localidade do banco de dados para que tenhamos suporte a nossa língua. Nesse caso utilizei o formato Portuguese [Brazil] com vemos na **Figura 6**.

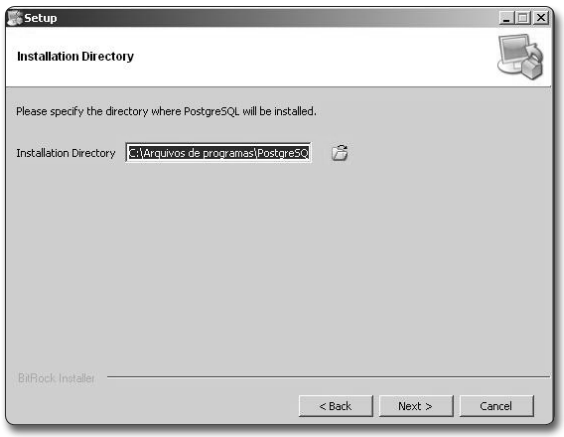


Figura 2. Diretório de instalação

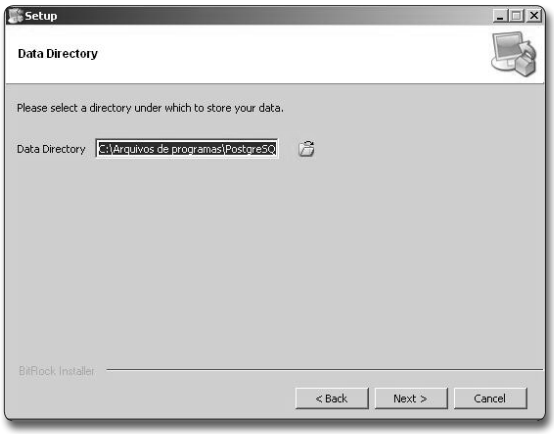


Figura 3. Diretório de banco de dados

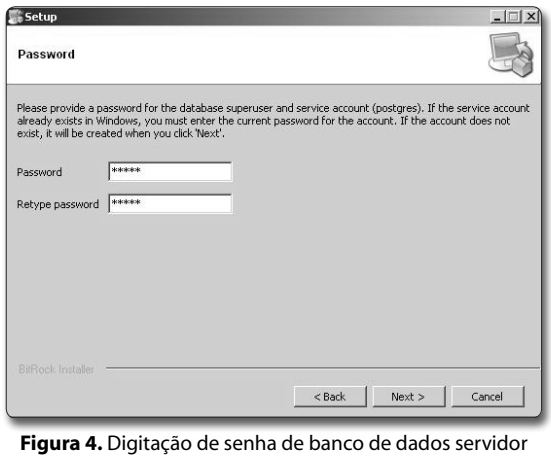


Figura 4. Digitação de senha de banco de dados servidor

Pronto, nas próximas telas bastará clicar no botão Install para que o instalador copie e instale todos os arquivos necessários para o uso do banco. Além do banco de dados, o PostGreSQL fará a instalação de um serviço no Windows para que o banco posso funcionar corretamente. O programa pgAminIII também será instalado. Ele é um client para acesso ao banco. Com ele podemos criar BD's, criar tabelas, fazer consultas, alterar as configurações de um banco e todas as configurações e ações necessárias para a manutenção de nosso SGBD. É semelhante ao IBExpert para Interbase/Firebird e MySQL Front ou HeidiSQL para MySQL.

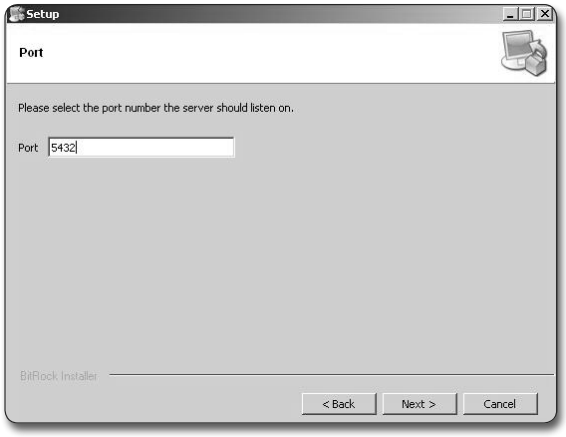


Figura 5. Porta de acesso ao banco

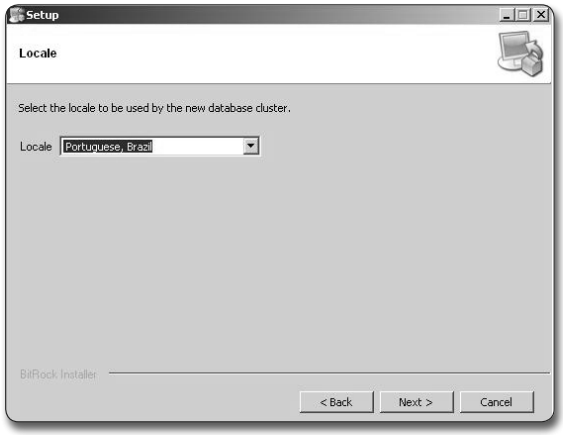


Figura 6. Configurando a localidade

Criando o banco de testes

Estando instalado o PostGreSQL, vamos criar um pequeno BD e entender algumas funcionalidades básicas do programa pgAdminIII. Acesse o menu Iniciar>Programas>PostGreSQL 8.3 e veja que diversos ícones foram adicionados conforme a **Figura 7**. Veja que temos Arquivos de Configuração, Documentação, Iniciar Serviço, Parar Serviço, pgAdmin III entre outros. Normalmente o serviço já é startado após a instalação, mas por via das dúvidas clique nessa opção para que o PostGreSQL entre no ar. Em seguida clique em pgAdmin III. O programa iniciará e uma tela como na **Figura 8** será mostrada.

Repare a esquerda que já temos um servidor instalado indicando a instalação do PG(PostGreSQL) foi efetuada com sucesso. Clicando duas vezes sobre PostgreSQL Database Server 8.3 (localhost:5432), todos os objetos do banco de servidor serão carregados. Obviamente que o servidor nos solicitará a senha de acesso e apenas temos que digitar a senha que configuramos durante a instalação.

Caso seja necessário adicionar um novo servidor PG, basta clicar em Arquivo>Adicionar ao Servidor. Na tela se abre, devemos informar alguns dados do novo servidor, tais como: Nome, Máquina, Porta e protocolo de acesso (SSL se houver), Manutenção do BD, Nome de usuário, Senha e Serviço. Veja a **Figura 9**. Como temos apenas um servidor instalado e configurado na máquina, sua configuração deve estar parecida com a **Figura 10**.

Abaixo do nó Bancos de Dados temos postgres, que é o banco

Figura 7. Programas do PostGreSQL

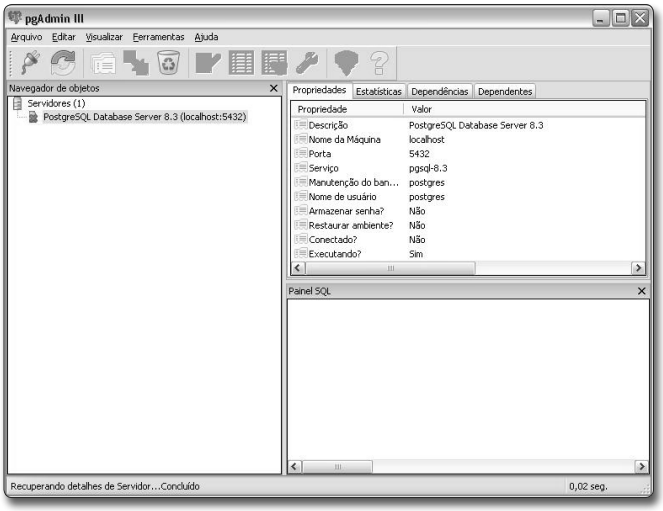


Figura 8. Tela inicial de pgAdmin III

Figura 9. Configuração de novo servidor

Figura 10. Configuração atual

de dados Demo instalado pelo próprio PG. Vamos criar nosso próprio BD clicando com o botão direito do mouse em Bancos de Dados e em seguida em Novo Banco de Dados. Na primeira tela apenas dê o nome de ClubeDelphi ao banco e selecione o Modelo postgres. Clique na aba Privilégios e marque All. Por fim selecione a aba SQL e note a instrução que o banco executará para criar o BD. Essas informações devem mudar de acordo com as configurações selecionadas na janela. Clique em Ok e confirme a criação do BD.

```
CREATE DATABASE "ClubeDelphi"
  WITH ENCODING='WIN1252'
       TEMPLATE=postgres;
```

Após a confirmação da criação de nosso banco, ele aparecerá na árvore logo a esquerda. Para conectar-se a ele basta clicar duas vezes. Verás que uma árvore de objetos será criada onde temos Catálogos, Esquemas e Replicação. Em esquemas ficam nossas tabelas. Abra essa opção e veja que temos os itens Domínios, Funções, Sequências, Tabelas, Funções de Gatilho e Visões. Sim, é esquisito ver tudo em português, mas não precisa esquentar a cabeça, basta escolher a versão em inglês ao instalar o software. Basicamente o que temos é:

- **Domains:** domínios do banco de dados, assim como já estamos acostumados em outros BD's;
- **Sequences:** Equivalente ao Generator ou Sequence no Firebird;
- **Tables:** nossas tabelas;
- **Triggers:** como já conhecemos em outros bancos;
- **Views:** As visões são as views, como o próprio nome sugere.

Vamos criar uma tabela bem simples apenas para entendermos como funciona. Clique com o botão direito do mouse em Tabelas e em seguida em Nova Tabela. Dê um nome a tela (Clientes) e clique em Colunas. Veja que é bem intuitivo e agradável o ambiente de criação dos objetos. Basta clicar no botão Adicionar e incluir os atributos do novo campo. Crie os campos Nome, Endereço, CEP, Cidade, Estado e Email. Veja no campo Tipo de Dado que já vários tipos não encontrados em outros BD's como Firebird, por exemplo. Para os campos texto utilize o tipo character e para campos inteiros, integer. Confirme a criação e note que a tabela já aparece a esquerda do software. Simples não?

Com um clique de direita na tabela recém criada, vemos que existem várias funcionalidades, tais como: cópia de segurança, scripts, manutenção etc. Muito bom, agora vamos inserir alguns registros para que possamos fazer testes. Clique com o botão direito na tabela e use o a opção Scripts>Insert. Na tela que se abre note que o pgAdmin III já cria uma instrução padrão, apenas preencha-a e execute clicando no botão Executar Consulta. Repita o procedimento quantas vezes forem necessárias. No meu caso executei a seguinte inserção:

```
INSERT INTO "Clientes"(
  "Nome", "Endereco", "CEP", "Email",
  "Telefone", "Cidade", "Estado",
  "ID")
VALUES (
  'Adriano Santos', 'Rua Hum, 1000', '08253470',
  'falecom@adrianosantos.pro.br',
  '11-555-5555', 'São Paulo', 'SP',
  1);
```

Criando uma aplicação

Bem, chegou a hora de fazermos uma aplicação simples capaz de trabalhar com o PostGreSQL através do Delphi. Antes de iniciamos, é importante sabermos que não existem drivers nativos para acesso ao PG dentro do Delphi, por isso necessitamos fazer uso de uma biblioteca de terceiros. A partir da versão 2007 do Delphi, a RAD Studio 2007, a suíte de componentes de acesso dbExpress foi totalmente reformulada transformau-se em um Framework e o melhor: totalmente desenvolvido em Delphi. A vantagem disso é que agora é possível, com um vasto conhecimento claro, desenvolver drivers para todo tipo de banco de dados. Possivelmente foi o que a empresa DevArt fez. Eles possuem drivers para diversos bancos incluindo o PG. Por isso, baixe uma versão Demo no site do fabricante e instale em seu computador. A instalação é rápida em simples. Acesse o link http://www.devart.com/ e baixei a versão Trial.

Após a instalação, crie uma nova aplicação Win32 no Delphi usando o menu File>New>VCL Forms Application – Delphi. No meu caso estou usando o RAD Studio 2009 para criar o exemplo. Localize na barra de componentes a paleta PostGreSQL Access. Observe na **Figura 11** que temos vários componentes de acesso ao PostGreSQL. Para o nosso exemplo, apenas adicione um TPgConnection mantendo seu nome default. Para configurar a conexão com o banco é bem simples, apenas selecione a propriedade Database e veja que o componente já exibe uma tela de entrada de senha. Entre com a senha do banco de dados e confirme. Feito isso a propriedade DataBase exibirá todos os bancos disponíveis bastando selecionar ClubeDelphi. Desmarque a propriedade Login prompt e pronto, tudo configurado.

Agora apenas adicione um componente TPgQuery ao formulário e clique duas vezes na propriedade SQL. Entre com a instrução que desejar, no meu caso joguei algo simples como:

```
SELECT * FROM CLIENTES
```

Repare que ao colocar o componente em tela, automaticamente ele já configurou a propriedade Connection apontando para o nosso TPgConnection. Agora adicione um TPgDataSource e ligue-o ao TPgQuery. Depois insira um DbNavigator e um DbGrid ligando-os a esse TPgDataSource. Por fim, insira um TPgUpdateSQL e ligue o PgQuery a ele. Se preferir adicione os campos da tabela ao FieldsEditor clicando duas vezes no TPgQuery. Não entrarei em detalhes por ser algo trivial no desenvolvimento. No evento OnCreate do formulário ordene a conexão com o banco de dados e abertura da query usando o código a seguir:

```
PgConnection1.Connected := True;
PgQuery1.Active := True;
```

Muito bom, execute a aplicação e veja que tudo está funcionando perfeitamente. Todas as operações de banco de dados funcionam exatamente como estamos acostumados em outros bancos de

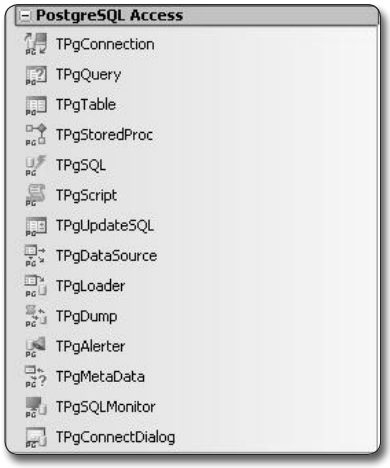


Figura 11. Paleta de componentes PG

dados e com outros componentes do tipo DataSet. Ainda podemos usar o conjunto de componentes para programação bi-direcional, ou seja, TPgConnection, TPgQuery, TDatasetProvider e TClientDataSet, mas isso já é assunto para outro artigo (veja um exemplo de tela em modo de construção na **Figura 12**).

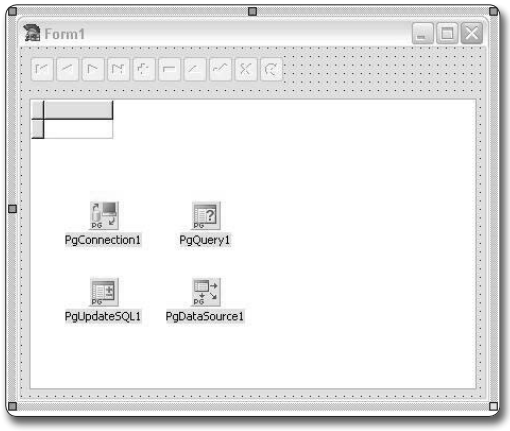


Figura 12. Layout da tela de testes

Conclusão

A FREE parece que pegou de jeito o mercado de TI. Temos praticamente todo tipo de solução free no mercado hoje em dia. O PostGreSQL é um exemplo disso. Mas FREE não é exemplo de má qualidade. O PostGreSQL é um excelente e robusto banco de dados que pode ser utilizado tranquilamente tanto para aplicações Desktop quanto Web.

Vale a pena testar e ver todo o seu potencial de perto.

Recomendo um estudo aprofundado do SGBD para então colocá-lo a prova em situações reais. Um forte abraço a todos e até a próxima. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

POO: Objetos Auto-Instanciáveis

Utilizando cláusulas opcionais da Unit para criar objetos dinamicamente

André Dias Martins
andredias.uem@gmail.com
É Bacharel em Sistemas de Informação pela Unipar - Universidade Paranaense, Campus Paranavaí. É Especialista em Desenvolvimento de Sistemas para Web, pela UEM Universidade Estadual de Maringá (2008). Aluno Especial do Programa de Mestrado em Ciência da Computação, pela UEM Universidade Estadual de Maringá (2009). Docente a 2 anos, atuando nos cursos de Ciência da Computação, e Sistemas de Informação. Tem experiência na área de Ciência da Computação, com ênfase em Desenvolvimento para Web, tendo especial interesse nas seguintes áreas: Engenharia de Software I, Engenharia de Software II, Engenharia de Software III, Lógica de Programação, Planejamento e Métricas de Software, Gestão de Projetos e Empreendedorismo.

Marcos Aurélio Rocha
marcosrocha85@gmail.com
Bacharel em Sistemas de Informação está atuando a mais de 5 anos com desenvolvimento de aplicações ERP e CRM em Delphi com banco de dados Firebird e SQLServer. Possui 2 anos de experiência em desenvolvimento de aplicações Web com PHP, Ajax e MySQL. Tem interesse em programação orientada a objeto no desenvolvimento de soluções práticas para aplicações comerciais, desenvolvimento de jogos em Delphi e OpenGL, Gerenciamento de Projetos e Segurança de Informações.

Resumo do DevMan

Neste artigo iremos verificar uma classe orientada a objetos e os principais problemas caso ela seja mal escrita. Iremos fazer um breve apanhado sobre o que ocorre na memória quando uma classe é instanciada e, por último mostrar a utilização prática das cláusulas initialization e finalization para automação de classes.

Nesse artigo veremos

- Criação simples de objetos e como funcionam;
- Utilização das cláusulas opcionais initialization e finalization.

Qual a finalidade

- Mostrar ao leitor como melhorar classes que podem ser auto-instanciáveis.

Quais situações utilizam esse recurso?

- Podemos utilizar este método para instanciar classes ou variáveis automaticamente no momento em que elas são necessárias recorrendo a duas cláusulas opcionais fornecidas pela IDE.

Entendendo a criação de Objetos e a problemática

Quando necessitamos instanciar um objeto é preciso fazer uma chamada ao método Create para que seja alocado o espaço necessário na memória e o objeto realmente existirá. No primeiro momento quando vamos instanciar um objeto na memória, o valor de sua variável é igual à nil, ou seja, nada. A partir do momento que é efetuada uma atribuição da chamada Create o programa associa a variável ao novo endereço de memória alocado que contém toda a classe instanciada. Na **Listagem 1** temos um exemplo de uma classe que chamamos de TMinhaClasse, salva em um arquivo chamado UMinhaClasse.pas, veremos a classe a seguir de forma mais detalhada:

Resumo do DevMan

A programação Orientação a Objetos é uma tecnologia que vêm sendo amplamente difundida e utilizada, pois a mesma proporciona maior facilidade e praticidade na manutenção de objetos em aplicações. Algumas vezes nos deparamos com procedimentos que precisam ser automatizados dispondo de classes para que haja maior rapidez e eficiência na manutenção, mas que não atendem às necessidades devido ao fato de precisarem ser instanciadas manualmente. Aqui veremos como instanciar classes para executar seus procedimentos e funções sem que um objeto esteja efetivamente instanciado na memória, entretanto sua criação torna-se praticamente automática apenas quando necessária.

Esta técnica é muito útil quando precisamos utilizar um objeto onde não há a certeza de que uma instância já se encontra alocada na memória, porém no momento do uso exige-se que ela seja instanciada para chamadas posteriores. Podemos dizer então que o objeto será dinamicamente criado no momento oportuno para que seja possível utilizar seus métodos e propriedades e ao fim ele é destruído e liberado o espaço de memória em que ele ocupava.

Listagem 1. Classe TMinhaClasse

```
01 unit UMinhaClasse;
02 interface
03 uses
04   Classes;
05 type
06   TMinhaClasse = class(TObject)
07   public
08     Lista: TStringList;
09     Nome: string;
10     constructor Create(AOwner: TObject);
11   end;
12 implementation
13 constructor TMinhaClasse.Create(AOwner: TObject);
14 begin
15   inherited Create;
16   Lista := TStringList.Create;
17 end;
```

Como podemos observar na **Listagem 1**, na **Linha 06** realizamos a declaração de nossa classe baseada em TObject, a seguir entre as **Linhas 08** e **09** estão definidas as variáveis públicas de classe, e em seguida na **Linha 10**, encontra-se a declaração do construtor padrão da classe. Logo na implementação (**Linha 15**) do construtor temos a chamada que invoca o construtor de TObject e inicializa a variável Lista de tipo TStringList (**Linha 16**). Repare que em nossa implementação não definimos o método destrutor. Neste caso, será utilizado o construtor da classe primária (TObject).

Nota do DevMan

Construtores em programação orientada a objeto nada mais são do que métodos responsáveis pela inicialização de uma instancia de objeto em sua chamada. Este é encarregado pela inicialização de variáveis e atributos de classe. O construtor padrão não necessariamente deve ser declarado, sendo assim, será adotado o construtor presente na super classe. Em algumas linguagens de programação o compilador automaticamente define um construtor padrão implícito.

Assim como o método responsável pela criação e instanciação dos atributos, existem os métodos denominados destrutores, responsáveis por realizar a liberação e destruição da classe, atributos e variáveis. Este por sua vez não necessariamente deve estar explícito, assumindo assim como o método construtor, o método destrutor padrão definido na super classe.

No caso do Delphi podemos utilizar o método Free ou FreeAndNil que faz automaticamente a destruição e liberação do ponteiro de memória.

ClubeDelphi PLUS www.devmedia.com.br/clubedelphi/portal.asp

Acesse agora mesmo o portal devmedia e confira um curso do autor Rodrigo Carreiro Mourão, onde o mesmo mostra como dominar as técnicas de programação orientada a objetos

http://www.devmedia.com.br/articles/viewcomp.asp?comp=13615

Na **Listagem 2** como iremos observar, temos a utilização da classe anteriormente criada onde após a chamada do construtor Create nossa variável MinhaVar terá todas as características de sua classe base e também seus objetos e variáveis com o conteúdo vazio, temos o exemplo de sua utilização em um novo formulário salvo como Unit1.pas.

Listagem 2. Utilização padrão de TMinhaClasse

```
// Adicionar a Unit UMinhaClasse da classe anteriormente criada
01 unit Unit1;

02 interface

03 uses
04   Windows, Forms, Classes, Controls, StdCtrls, Dialogs, UMinhaClasse;

05 type
06   TForm1 = class(TForm)
07     Button1: TButton;
08     procedure Button1Click(Sender: TObject);
09   private
10     { Private declarations }
11   public
12     { Public declarations }
13   end;

14 var
15   Form1: TForm1;
16   MinhaVar: TMinhaClasse;

17 implementation

18 procedure TForm1.Button1Click(Sender: TObject);
19 begin
20   MinhaVar := TMinhaClasse.Create(Self);
21   MinhaVar.Nome := 'JOÃO JOSÉ DA SILVA';
22   MinhaVar.Lista.Add('IDADE: 29 ANOS');
23   ShowMessage(MinhaVar.Nome +#13+ MinhaVar.Lista[0]);
24 end;
```

Note que no código presente na **Listagem 2** simplesmente criamos um novo formulário e adicionamos um botão (**Button1**). Repare que inicialmente declaramos à nossa sessão uses a Unit referente à nossa classe (**Linha 04**). A partir da **Linha 14**, onde temos a definição da variável referente ao próprio Form, fazemos referência à nossa classe (TMinhaClasse) na qual denominamos como MinhaVar. Logo na **Linha 18**, temos a implementação do evento OnClick do botão. A variável simplesmente tem sua criação através do método Create (**Linha 20**). Em seguida nas **Linhas 21** e **22**, o conteúdo é passado às variáveis Nome e Lista. Para que finalmente sejam exibidos através de uma mensagem conforme mostra a **Linha 23**.

Nota

Escrever uma classe sem atenção pode acarretar em erros de violação de acesso à memória (Access Violation) ao tentar acessar uma de suas propriedades. Isto ocorre pelo fato de que no momento em que se tenta associar algo à variável Nome, o objeto na qual ela pertence ainda não foi instanciado, pois somente ao passar pelo end que ela passará a existir.

O método automático para instanciar classes

Mesmo na primeira versão do Delphi, havia um meio de adicionar códigos a uma Unit através do método AddExitProc() porém, a partir da versão 2, foram incorporadas as cláusulas

opcionais initialization e finalization para que fosse possível executar procedimentos antes e depois da inicialização e finalização da Unit pelo programa. Quando uma parte do código necessita efetuar chamadas a uma Unit qualquer, o programa verifica se existe a cláusula initialization para que sejam executados os procedimentos antes que seja completada a chamada. Adicionando um código para instanciar nossa classe nessa cláusula estamos criando um mecanismo automático para que a variável possa ser utilizada sem que seja necessário digitar o código de criação dela em um DataModule, por exemplo.

A seguir na **Listagem 3**, faremos algumas modificações em nossa Unit, incluindo os códigos de inicialização e finalização para que o procedimento de auto instanciação e destruição sejam realizados.

Listagem 3. Unit de TMinhaClasse com cláusulas initialization e finalization

```
// Adicionar SysUtils e a Unit1 ao uses da classe
01 uses
02   Classes, SysUtils;

03 type
04   TMinhaClasse = class(TObject)
05   public
06     Lista: TStringList;
07     Nome: string;
08     constructor Create(AOwner: TObject);
09   end;
10 implementation
11 uses Unit1;
12 constructor TMinhaClasse.Create(AOwner: TObject);
13 begin
14   inherited Create;
15   Lista := TStringList.Create;
16 end;
17 initialization
18   if not Assigned(MinhaVar) then
19     MinhaVar := TMinhaClasse.Create(nil);
20 finalization
21   if Assigned(MinhaVar) then
22     FreeAndNil(MinhaVar);
```

Podemos ver na **Listagem 3** como fica a Unit UMinhaClasse.pas com as cláusulas initialization e finalization implementadas a fim de que nossa classe seja instanciada apenas uma vez ao ser necessário e destruída quando seu uso seja dispensado. Na **Linha 17** temos a definição da cláusula initialization, na qual fará que seja considerado primeiramente o código ali contido. O teste feito na **Linha 21** verifica se não há conteúdo dentro da variável MinhaVar (definida no Form1), sendo verdadeira esta condição, a variável será automaticamente instanciada a partir da classe (**Linha 19**). Na **Linha 20** é definida a cláusula finalization. A **Linha 21** verifica se existe se a variável MinhaVar não está nula, para que seja destruída e liberada da memória como mostrado na **Linha 22**. Repare que as verificações que certificam que nossa classe já foi instanciada em initialization e finalizada em finalization, encarregam-se de evitar erros de memória, note também que não há necessidade de separar os códigos em blocos com begin e end, portanto todo código colocado entre estas cláusulas somente precisam possuir blocos em caso de um if, for ou while com mais de uma linha de execução como utilizamos normalmente em nossos códigos.

Após a classe TMinhaClasse ter sido devidamente escrita com seus códigos de inicialização e finalização podemos utilizá-la com a certeza de que quando for necessária ela estará lá. Veremos na **Listagem 4** como proceder com os ajustes para que possamos realizar o teste prático final.

Listagem 4. Utilização da classe TMinhaClasse automática

```
01 uses
02   Windows, Forms, Classes, Controls, StdCtrls, Dialogs, UMinhaClasse;
03 type
04   TForm1 = class(TForm)
05     Button1: TButton;
06     procedure Button1Click(Sender: TObject);
07   private
08     { Private declarations }
09   public
10     { Public declarations }
11   end;
12 var
13   Form1: TForm1;
14   MinhaVar: TMinhaClasse;
15 implementation
16 procedure TForm1.Button1Click(Sender: TObject);
17 begin
18   MinhaVar.Nome := 'JOÃO JOSÉ DA SILVA';
19   MinhaVar.Lista.Add('IDADE: 29 ANOS');
20   ShowMessage(MinhaVar.Nome+#13+MinhaVar.Lista[0]);
21 end;
```

Observando a **Listagem 4**, podemos constatar que no código presente no evento OnClick (a partir da **Linha 16**) ouve uma pequena modificação, na qual não é realizada a declaração explícita de criação do objeto MinhaVar, este código por sua vez, não acarretará em erros, visto que a diretiva initialization adicionada anteriormente à nossa classe, se encarregará de realizar a criação como pudemos ver de forma detalhada na **Listagem 3**. Neste caso, clicando no Button teremos o resultado exibido na **Figura 1**.

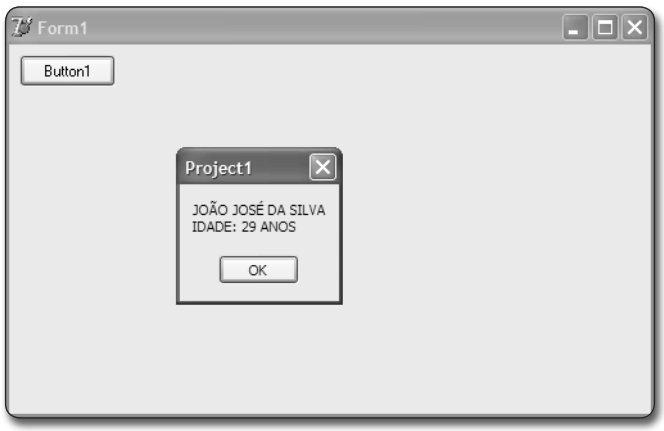


Figura 1. Mostrando valores passados para a classe que foi criada automaticamente

Vale salientar que após o encerramento do bloco de instruções na qual referencia a classe, a própria Unit acionará o método finalization fazendo com que a variável MinhaVar seja liberada da memória.

Nota

A utilização das cláusulas initialization e finalization deve ser feita com muito cuidado, pois o programa leva em consideração as Units na ordem em que foram declaradas executando os códigos contidos nestas cláusulas. Portanto tenha cuidado ao alterar a ordem de declaração de suas Unit's caso contrário você poderá ter erros de difícil manutenção.

Implementando um DataModule de teste

Para finalizarmos nosso artigo, iremos criar um exemplo de aplicação das cláusulas initialization e finalization em um DataModule, mais uma vez mostrando uma classe auto-suficiente, este tipo de caso é muito útil no cotidiano. Selecione o menu File>New>Data Module e altere seu Name para DMTeste, salve-o com o nome de UDMTeste.pas para que possamos implementar o código de criação automática, conforme mostra a **Listagem 5**.

Listagem 5. Implementação da automatização em um Data Module

```
01 unit UDMTeste;
02 interface
03 uses
04   SysUtils, Classes;
05 type
06   TDMTeste = class(TDataModule)
07   private
08     { Private declarations }
09   public
10     { Public declarations }
11   end;
12 var
13   DMTeste: TDMTeste;
14 implementation
15 {$R *.dfm}
16 initialization
17   if not Assigned(DMTeste) then
18     DMTeste := TDMTeste.Create(nil);
19 finalization
20   if Assigned(DMTeste) then
21     DMTeste := nil;

22 end.
```

Observando a nossa implementação, vemos que as modificações apenas foram inseridas a partir da **Linha 16**, e assim como antes, o mesmo teste foi realizado. Neste caso conservamos a variável padrão criada pelo Delphi, que refere-se ao próprio Data Module (**Linhas 12** e **13**). Novamente na **Linha 17** verificamos se a variável DMTeste é nula através da negação do método Assigned, sendo assim, realizamos a criação da instância (**Linha 18**). Na **Linha 19** temos a finalização. Repare que o mesmo teste foi feito, porém inversamente, neste caso se houver conteúdo (**Linha 20**), então a variável DMTeste recebe o conteúdo nil. Uma importante observação a ser feita, é que não podemos utilizar o método FreeAndNil neste caso. Isso se deve ao fato de que os Data Module's sempre são criados primeiramente, e destruídos por último na aplicação, então neste caso apenas apontamos a variável para nulo, para que o objeto seja realmente destruído no término da aplicação. Uma forma de comprovar isto é realizando a depuração do projeto.

Conclusão

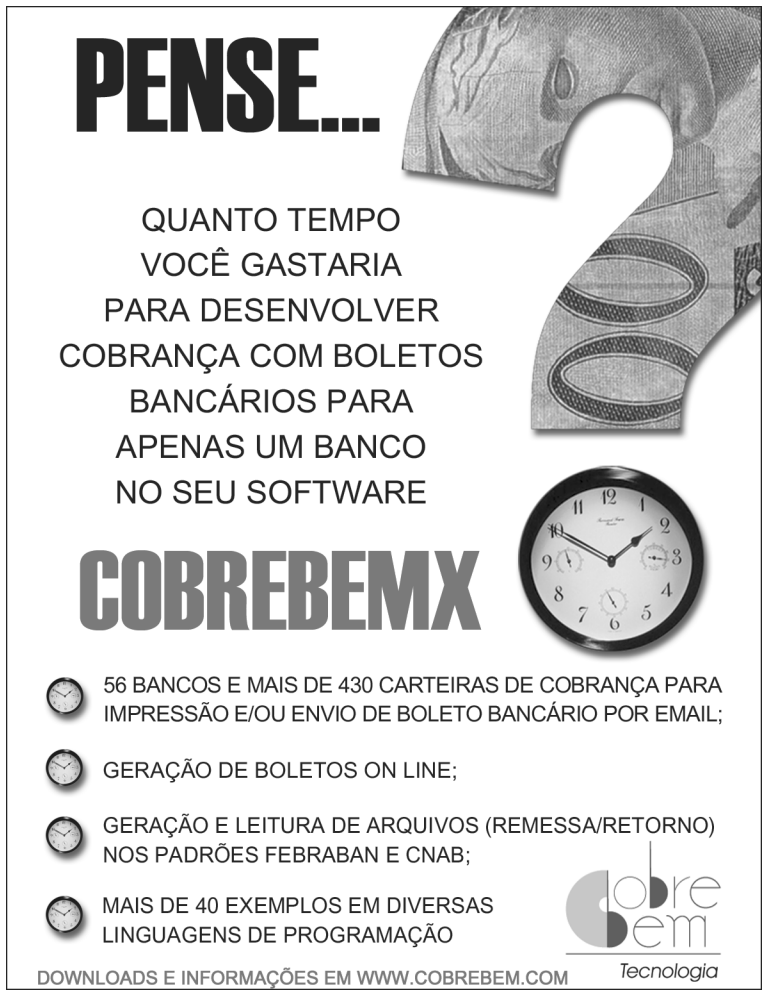
Com estas definições, conclui-se que em determinadas situações rotineiras a utilização de uma classe auto-instanciável garante funcionamento eficaz, evitando a manipulação manual da instância de um objeto, uma vez que utilizamos suas propriedades e métodos sem que este esteja efetivamente instanciado na memória. As aplicações deste método são muito utilizadas em componentes e interfaces onde há extrema necessidade de que sejam inicializados e/ou finalizados diversos dados antes de seu uso. Objetivando com isto, maior praticidade, e maior satisfação no que se refere à busca de melhores resultados. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback



Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Curso de Delphi Prism – Parte 2

Novidades da Linguagem – Métodos Anônimos e Delegates

Guinther Pauli
guinther@devmedia.com.br
Tem 21 anos de experiência na área de desenvolvimento e arquitetura de software, dos quais 14 dedicados ao Delphi e 9 ao .NET. Certificado Borland em Delphi 6, 7, 2005, 2006, Delphi para Web e Kylix, Certificado Microsoft MCP, MCAD, MCSD. NET, MCTS e MCPD. Editor geral da Revista .NET Magazine Brasil e professor de Pós-Graduação. Publicou mais de 150 artigos, 400 vídeo-aulas, palestrante em 5 edições da Borland Conference. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas.

Resumo do DevMan

Continuando o curso de Delphi Prism, neste artigo vamos estudar o que são métodos anônimos, recurso incluído no C# 2.0 e agora disponível para serem usados por desenvolvedores Delphi Prism. Veremos também o que são delegates, recurso do C# e .NET, presente também no Prism (e no Delphi Win32, mas não com esse nome). Para isso, vamos voltar ao passado e ver como as coisas funcionavam no Delphi 7 Win32, entendendo facilmente o porquê de existirem esses recursos na nova versão do Delphi for .NET.

Na primeira parte deste curso, vimos um pouco da história do suporte do Delphi ao .NET Framework. Examinamos o que considero serem as principais novidades da linguagem do Prism, como variáveis in-line (onde você pode declarar variáveis e objetos dentro do código, e não só na sessão var) e também o recurso de type-inference, onde é possível omitir o tipo da variável, usando mesmo assim, objetos fortemente tipados. Também

Nesse artigo veremos

Lançado pela Embarcadero, empresa que adquiriu a CodeGear, o Delphi Prism é a nova forma de desenvolver aplicações para o .NET Framework e Mono utilizando o Object Pascal e o IDE do Visual Studio 2008. Nesta segunda parte do curso, apresento mais sobre os novos recursos da linguagem utilizada pelo Delphi Prism, que são inúmeros se compararmos com o Delphi for .NET tradicional ou mesmo Delphi Win32.

Qual a finalidade

Comparar a sintaxe do antigo Pascal e Delphi tradicional com a moderna e poderosa linguagem suportada pelo Delphi Prism, conhecer os principais novos recursos da linguagem e vantagens.

Quais situações utilizam esse recurso?

O Delphi Prism abre uma porta incrível para os desenvolvedores Delphi, pois podem agora utilizar um IDE robusto, rápido e estável para construir aplicações para .NET, incluindo as novas tecnologias ASP.NET 3.5, ASP.NET AJAX, WPF, WCF, MONO etc. usando uma versão evoluída do Object Pascal, que mescla os melhores recursos do Delphi e do C# 3.0.

vimos que é possível inicializar variáveis locais e usar type-inference na sessão var, sem recorrer a recursos não muito atraentes à programação orientada a objetos.

Neste artigo, vamos conhecer os métodos anônimos, que como o próprio nome do recurso sugere, são métodos que não têm nome. Esse é um recurso que possui uma íntima relação com eventos (e delegates, visto também neste artigo). Novamente peço para que você abra sua mente, deixe de ser um seguidor fiel do Niklaus Wirth e dê um passo rumo ao futuro.

Métodos Anônimos

Como de costume, para entendermos o presente e futuro, vamos analisar o passado, como funcionavam as coisas no mundo Delphi 7 Win32.

Em um formulário TForm do Delphi, quando adicionamos um manipulador para um evento de um controle de tela (Button, Edit, DBGrid), ou mesmo um componente não-visual (ClientDataSet, Timer, SQLConnection), estamos nada mais nada menos que apontando uma propriedade para um método. De fato, um evento é ponteiro para um método de uma classe, e também tem um tipo, como TNotifyEvent, principal tipo de evento da VCL do Delphi.

Por exemplo, na classe TControl, encontramos a definição do evento OnClick, que serve de base para muitas outras classes que herdam o evento, como um TButton. Observe o código da **Listagem 1**. A declaração do evento funciona exatamente como uma propriedade normal, possui uma variável privada que é mapeada por métodos Get e Set. Só que, ao invés de atribuirmos um valor para essa propriedade, como um objeto, uma string, um inteiro, apontamos para um método, que no Win32, obrigatoriamente deve ser um método de uma classe, functions rotinas e procedures fora desse escopo não funcionam (no Delphi for .NET do BDS funciona, pois units viram classes nos bastidores).

Listagem 1. Declaração do evento OnClick em TControl

```
TControl = class(TComponent)
...
FOnClick: TNotifyEvent;
...
property OnClick: TNotifyEvent
read FOnClick write FOnClick stored IsOnClickStored;
```

Se colocarmos um Button em um formulário e dermos um duplo clique nele, o IDE vai automaticamente criar um manipulador para o seu evento padrão, que no caso de um botão, é o OnClick. Então, implementamos o método. O código para isso pode ser observado na **Listagem 2**. Coloquei nessa listagem também a declaração da classe na sessão Type. Veja que o método Button1Click é declarado logo abaixo de TForm, depois do Button, e não possui um especificador de visibilidade como public ou private. Como TForm descende de TPersistent, classe que ativa a diretiva {$M+}, o compilador ativa a RTTI (Runtime type Information) e consegue ler do arquivo DFM a atribuição do evento antes de chamar o importante método virtual Loaded de TComponent. Por isso, por padrão, classes que descendem de TPersistent e não declaram o especificador para um método ou objeto (como no exemplo), automaticamente usam o modificador published para esses membros. Esse é um prerequisito para um método funcionar, experimente mover Button1Click para o private e você verá o que estou querendo dizer. Na **Listagem 3**, veja o fragmento de código do DFM que representa essa atribuição.

Listagem 2. Funcionamento de um evento

```
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowMessage('Guinther');
end;
```

Listagem 3. Evento atribuido no DFM

```
  object Button1: TButton
    Left = 128
    Top = 72
    Width = 75
    Height = 25
    Caption = 'Button1'
    TabOrder = 0
    OnClick = Button1Click
  end
```

Podemos também atribuir esse evento em tempo de execução, normalmente, como se fosse uma propriedade qualquer. Experimente retirar o código do DFM que aponta o OnClick para o Button1Click (ou limpe o valor da propriedade OnClick no Object Inspector) e no OnCreate do formulário digite o código da **Listagem 4**. A implementação do método, nesse caso Button1Click, além de ser published, tem que bater com o tipo do evento esperado, nesse caso, TNotityEvent. Eventos desse tipo sempre esperam um parâmetro Sender: TObject, que é o objeto que sofreu o evento.

Listagem 4. Evento atribuído em runtime

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Button1.OnClick := Button1Click;
end;
```

E no Delphi Prism? Podemos fazer exatamente a mesma coisa no Prism, funciona exatamente da mesma forma, com algumas poucas diferenças. Se colocarmos um Button em um Web Form e dermos um duplo clique, vamos ter o código da **Listagem 5**. Aqui, o método Button1_Click não é published, é protected. Não existe no .NET Framework o conceito de

published, pois para usar RTTI em um objeto, fazemos uso de Reflection. Além disso, o método recebe dois parâmetros, além de System.Object (o TObject do .NET), recebe também argumentos na variável e, que variam de acordo com o controle. Na **Listagem 6** vemos o trecho de código ASPX que vincula o evento ao método.

Listagem 5. Eventos no Delphi Prism - .NET

```
type
  _Default = public partial class(System.Web.UI.Page)
  protected
    method Button1_Click(sender: System.Object; e: System.
     EventArgs);
    method Page_Load(sender: Object; e: EventArgs);
  end;

implementation

method _Default.Page_Load(sender: Object; e: EventArgs);
begin

end;

method _Default.Button1_Click(sender: System.Object; e: System.
EventArgs);
begin
    Response.Write("Guinther");
end;
```

Listagem 6. Código ASPX

```
<asp:Button ID="Button1" runat="server"
 onclick="Button1_Click"
Text="Button" />
```

Nota do DevMan

Você notou na **Listagem 5** que o Button não consta na declaração da classe, como na VCL Win32, apenas no arquivo ASPX (o que seria equivalente ao DFM da VCL)? Desde o .NET 2.0 existe o recurso de classes parciais (estudadas mais adiante neste curso), onde o compilador "unifica" o código da classe da página ASPX com o código da classe do code-behind, usando uma mescla de recursos com herança e classes parciais. Com isso, a classe no PAS enxerga os controles da classe ASPX, que podem ser acessados normalmente via código. Com isso, não há a necessidade de termos a declaração de um Button, por exemplo, no ASPX e no PAS ao mesmo tempo, algo que não pode ser feito na VCL (PAS com DFM).

Em tempo de execução, se quisermos atribuir o evento para apontar para um método, podemos fazer como na **Listagem 7** (retire a atribuição do ASPX que atribui o evento do botão), como no Delphi Win32. Ao clicar no botão, temos o mesmo resultado se comparado ao procedimento feito em tempo de design.

Listagem 7. Atribuindo um evento no .NET

```
method _Default.Page_Load(sender: Object; e: EventArgs);
begin
    Button1.Click += Button1_Click;
end;

method _Default.Button1_Click(sender: System.Object; e: System.
EventArgs);
begin
    Response.Write("Guinther");
end;
```

Reparou algo estranho na **Listagem 7**? Olhe bem. O operador que usamos para vincular o método manipulador de evento ao evento Click é +=, e não := (o que equivale ao include em antigas implementações do Delphi for .NET). Isso acontece porque no .NET Framework podemos ter um evento que chama vários métodos ao ser clicado (multi-cast events), quebrando uma limitação que existe na VCL. Parece estranho? Se você ainda estiver com a mente fechada, com certeza sim. É perfeitamente possível escrever o código da **Listagem 8**. Quando o botão for clicado, em um único evento, três métodos serão chamados.

Listagem 8. Multi-cast events

```
_Default = public partial class(System.Web.UI.Page)
  protected
    method Button_Click_1(sender: System.Object; e: System.EventArgs);
    method Button_Click_2(sender: System.Object; e: System.EventArgs);
    method Button_Click_3(sender: System.Object; e: System.EventArgs);
    method Page_Load(sender: Object; e: EventArgs);
  end;

implementation

method _Default.Page_Load(sender: Object; e: EventArgs);
begin
    Button1.Click += Button_Click_1;
    Button1.Click += Button_Click_2;
    Button1.Click += Button_Click_3;
end;

method _Default.Button_Click_1(sender: System.Object; e: System.EventArgs);
begin
    Response.Write("Guinther1");
end;

method _Default.Button_Click_2(sender: System.Object; e: System.EventArgs);
begin
    Response.Write("Guinther2");
end;

method _Default.Button_Click_3(sender: System.Object; e: System.EventArgs);
begin
    Response.Write("Guinther3");
end;
```

E onde entra o anonimato? Os métodos anônimos apareceram no .NET com o C# 2.0, e no Prism eles também já podem ser usados. Juro que à primeira vista que, como eu, sendo um OO purista mas não cético, você vai achar muito estranho. Analise o código da **Listagem 9**, que faz a mesma coisa que o código da Listagem 8. Sim, eu não apontei o evento para um método, mas para um bloco de código! In-line, sem declaração de cabeçalho, nem declaração na classe (uma limitação que o Delphi ainda tem comparado ao C#), sem mesmo nome, enfim, anônimo. De praxe, vimos que dá para usar o anonimato com mult-cast events.

Muitos estão se perguntando agora o porquê de utilizar esse tipo de implementação, que convenhamos, é confusa e não muito legível, principalmente se você estiver trabalhando em um código que será também trabalhado com outros desenvolvedores. A vantagem é óbvia: produtividade. Você não precisa ficar declarando dezenas de métodos na interface (ainda mais que estamos falando do Delphi), implementado-os, refatorando-os, vai trabalhando conforme for necessário. Precisa que um evento seja manipulado por um método? Implemente-o na mesma hora, in-line. Como vemos, existem prós e contras.

Nota do DevMan

Na VCL, por padrão, a maioria dos eventos leva o nome OnXXXXXX (OnClick, OnChange, OnCreate). Algumas exceções, são os que começam com BeforeXXXXX, AfterXXXXX, como eventos de TDataSet e descendentes. Esses eventos, na implementação interna da VCL, verificam se o campo privado de um tipo evento está apontando para um método, e então o chamam. Isso é feito em um método de despacho normalmente com o a diretiva virtual, para que você possa usar polimorfismo. Por exemplo, TControl declara o método virtual Click, que verifica se FOnClick aponta para algo, e o chama. Algo como mostrado abaixo:

```
procedure Click; dynamic;
...
procedure TControl.Click;
begin
  …
  if Assigned(FOnClick) then
    FOnClick(Self);
end;
```

Eu falei virtual, mas repare que o método Click é dynamic. Isso surgiu no Delphi 4, é a mesma coisa que virtual, porém instrui o compilador a gerar uma VMT (Virtual Method Table) mais otimizada, especialmente para cadeias de classes com grande hierarquia (muitos ancestrais).

Ou seja, por exemplo, se você vai criar uma nova classe chamada TButtonSound que emite um "bip" toda vez que clicada, não vai atribuir o seu evento OnClick, pois você está escrevendo uma classe e não usando uma. Nesse caso, sobrescreva Click com override. Assim, quem usar seu novo controle, poderá usar normalmente o OnClick.

No .NET, a nomenclatura é exatamente o inverso. Ou seja, o evento se chama XXXXX, e o método de despacho virtual interno à classe se chama OnXXXXX. Por exemplo, o evento Click é testado e chamado pelo método virtual interno OnClick.

Listagem 9. Métodos anônimos em Delphi

```
type
  _Default = public partial class(System.Web.UI.Page)
  protected
    method Page_Load(sender: Object; e: EventArgs);
  end;

implementation

method _Default.Page_Load(sender: Object; e: EventArgs);
begin
    Button1.Click +=
        method
        begin
            Response.Write("Guinther1");
        end;
    Button1.Click +=
        method
        begin
            Response.Write("Guinther2");
        end;
    Button1.Click +=
        method
        begin
            Response.Write("Guinther3");
        end;
end;
```

Veja bem, não pense pequeno, não pense em botões, pense que você pode usar esse recurso em qualquer evento de qualquer classe do .NET Framework, que são milhares. A propósito, a implementação em C# para o código Delphi Prism da **Listagem 9** é mostrada na Listagem 10. Mais adiante veremos o que é um delegate, que usei no código.

Listagem 10. Métodos anônimos em C#

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Button1.Click += delegate
        {
            Response.Write("Guinther1");
        };
        Button1.Click += delegate
        {
            Response.Write("Guinther2");
        };
        Button1.Click += delegate
        {
            Response.Write("Guinther3");
        };
    }
}
```

Nota

Ao longo deste curso, vou fazer várias referências ao C#. Mesmo usando o Delphi Prism, é importantíssimo que você conheça a semântica da linguagem C#, pois ao desenvolver para .NET, toda documentação, livro, artigo e outros recursos que você for utilizar, provavelmente será escrito em C# ou VB.NET. Mas não se preocupe, é muito simples identificar a diferença entre um {} e begin end!

De volta ao passado

Para quem já criou componentes e precisou declarar um evento que recebesse parâmetros que não o simples Sender: TObject, precisou fazer algo como na **Listagem 11**. Aqui simulei uma nova classe, fictícia, que representa uma imagem. Ao ser clicada, ela dispara um evento que passa as coordenadas X e Y do clique como parâmetros, para que possam ser manipuladas pelo programador que usar a classe. Na sessão type, declaramos um novo tipo de procedure (usando a sintaxe procedure of object), que chamei de TXYClickEvent (lembra do TNotifyEvent?). A vantagem é que nosso evento reconhece as coordenadas do clique e passa essa informação ao usuário do componente. A classe, usando uma boa prática que nosso amigo DevMan já comentou, possui um método de despacho virtual que aciona o OnXYClick. Uma possível implementação para o uso da classe é mostrada na **Listagem 12**.

Boa Prática

Declare tipos específicos de evento com o sufixo Event, como fiz na **Listagem 10**.

Listagem 11. Procedure of Object e evento customizado

```
type
  TXYClickEvent = procedure (Sender: TObject; X,Y: integer) of
object;

  TClickImage = class(TImage)
  private
    FOnXYClick: TXYClickEvent;
  protected
    procedure XYClick (Sender: TObject; X,Y: integer); virtual;
  public
    property OnXYClick: TXYClickEvent read FOnXYClick write
FOnXYClick;
  end;
...
procedure TClickImage.XYClick(Sender: TObject; X,Y: integer);
begin
  if Assigned(FOnXYClick) then
    FOnXYClick(self,X,Y);
end;
```

Listagem 12. Usando a classe com evento customizado

```
procedure TForm1.ImagemClicada(Sender: TObject; X,Y: integer);
begin
    ShowMessage(Format('Imagem clicada em %d,%d',[x,y]));
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  Pintura: TClickImage;
begin
  Pintura := TClickImage.Create(self);
  Pintura.OnXYClick := ImagemClicada;
  // simula evento de click
  Pintura.OnXYClick(self,20,30);
end;
```

Nota

Como comentei, o exemplo é fictício e tem fins didáticos. Já existe um evento customizado na VCL que captura um click repassando os valores das coordenadas X e Y, usada no evento OnMouseDown por exemplo. Ele se chama TMouseEvent, declarado na unit Controls.

Delegates: Voltando ao presente

Ok, se eu tivesse falado o que era delegate no início do artigo, muitos talvez não entenderiam. Esse é um conceito do .NET, que na verdade não é nada novo para os desenvolvedores Delphi. Um delegate é usado para "definir um tipo customizado de evento". Bingo! Então, um delegate é o nosso velho conhecido procedure of object (**Listagem 11**). Prepare-se, se está começando agora no .NET, você vai ouvir muito esse termo. Aliás, delegate costuma ser confundido com o próprio evento, e não o tipo.

No Delphi Prism, para entrar em conformidade com a CLS (Common Language Specification – veja nota do DevMan um pouco sobre as siglas usadas no .NET), foi implementada a nova palavra-reservada delegate. Então, nosso código da **Listagem 11**, portado para o Prism, ficaria como na **Listagem 13**. Ao invés de usar o procedure of object, usamos a keyword delegate. Na declaração da classe, simplificada, a também nova palavra reservada event identifica que XYClick (sem o "On") é um evento (não use mais a palavra property). E de praxe, novamente, a keyword delegate agora usada em sua segunda função, designando qual o membro privado é a variável do tipo do nosso evento. O raise não é para levantar uma exceção não, ele instrui o compilador a gerar código para que possamos chamar diretamente nosso evento a partir de outra classe (isso é feito na **Listagem 14**).

A **Listagem 14** mostra um possível uso para a classe ClickImage. Como já estudamos métodos anônimos, nada melhor que usá-los para testar nosso delegate. Usamos também type inference, estudado no artigo anterior. Interessante, veja que o método é anônimo recebe parâmetros. Sim, também é possível. Se declarássemos os parâmetros de forma diferente, a assinatura do método anônimo não iria casar com a assinatura declarada pelo delegate (sender, x e y), gerando um erro.

Listagem 13. Delegates em Delphi Prism

```
type

  XYClickEvent = delegate (Sender: Object; X,Y: integer);

  ClickImage = class(Image)
  private
    FXYClick: XYClickEvent;
  public
    event XYClick: XYClickEvent delegate FXYClick raise;
  end;
```

Nota do DevMan - Glossário.NET

CLI – Common Language Infrastructure: é um dos principais fundamentos da plataforma .NET. CLI é uma especificação que define um ambiente virtual de execução (VM – virtual machine), desenhado para suportar diversas linguagens de programação que podem interagir entre si. A Microsoft submeteu essa especificação à ECMA (European Computer Manufacturer's Association – www.ecma.ch) para que outras empresas pudessem oferecer diferentes implementações. A especificação CLI inclui também as especificações do CTS, CLS, MetaData, CIL e VES.

CTS – Common Type System – especificação que define um sistema comum de tipos (valor ou referência) que podem ser utilizados em diferentes linguagens de programação.

CLS – Common Language Specification – subconjunto do CTS, define convenções e regras destinados a garantir a interoperabilidade entre diferentes linguagens (C#, Delphi, VB etc), incluindo herança, manipulação de exceções e polimorfismo.

CLR – Common Language RunTime – implementação da Microsoft para a especificação CLI. Para aqueles que conhecem o Java, o CLR é muito semelhante à JVM (máquina virtual Java). Entre suas principais funções, estão: conversão do código IL (Intermediate Language – linguagem intermediária) em código nativo da plataforma, execução das aplicações .NET, gerenciamento de memória automático (Garbage Collection), e verificações de segurança.

CIL – Common Intermediate Language – compiladores construídos para a plataforma .NET não geram instruções nativas do processador. Ao invés disso, geram código intermediário (IL – Intermediate Language). Novamente, se você conhece Java, pense nisso como algo semelhante ao bytecode, o que permite que, pelo menos em teoria, seu código compilado possa rodar futuramente em diferentes plataformas.

JIT – Just-In-Time Compiler – mecanismo associado ao CLR que permite a compilação do código IL em código nativo da plataforma no momento da sua execução.

FCL – Framework Class Library – conjunto de classes do .NET Framework (o FCL é para o .NET Framework o que a VCL é para o Delphi).

Listagem 14. Usando a classe com delegate

```
method _Default.Page_Load(sender: Object; e: EventArgs);
begin
    var Pintura := new ClickImage();
    Pintura.XYClick += method // eu não tenho nome :(
        (Sender: Object; X,Y: Integer)
    begin
        Response.Write(string.Format(
            "Imagem clicada em {0},{1}",X,Y));
    end;
    { Simula evento de click, se não fosse
      o raise declarado no event, não seria
      possível chamar o evento diretamente }
    Pintura.XYClick(self,20,30);
end;
```

Nota

Em ambos os exemplos, Win32 e Prism, simulamos a ocorrência do evento, pois obviamente não instalamos nossa classe no IDE nem incluímos a lógica para interceptar o Click em X,Y. O real objetivo deste artigo está longe disso, aqui estamos focando na linguagem. Dessa forma, chamamos o método diretamente pela sua referência, o Delegate. A título de curiosidade, no ASP.NET existe um controle que permite o uso de técnicas relacionadas a cliques localizados, o ImageMap.

Conclusão

Como vimos, um delegate é um tipo por referência usado para referenciar um método. Qualquer método que satisfaça a assinatura do delegate, que consiste do tipo de retorno e dos tipos dos parâmetros, pode ser atribuído a uma instância do mesmo. A criação de métodos anônimos é essencialmente uma maneira de passar um bloco de código para um parâmetro delegate.

Com isso, finalizamos mais essa parte do nosso curso. Com este artigo, aprendemos não só as novidades do Delphi Prism com linguagem, mas também recursos antigos da VCL, do Delphi Win32, do .NET Framework como um todo, e por que não dizer, aprendemos C#. Reforçando, Delphi Prism para mim é "D#". Não se surpreenda se em breve você puder escrever this no lugar de self e o compilador não reclamar!

Na próxima edição vamos estudar tipos anônimos / implícitos. Abaixo, vou deixar um bloco de código para você já ir pensando e analisando, e principalmente, liberando-se de alguns vícios e abrindo a mente para o futuro:

```
var Eu := new class (Nome := "Guinther", Idade := 31);
```

Grande abraço e até a próxima edição. ●

Dê seu feedback sobre esta edição!

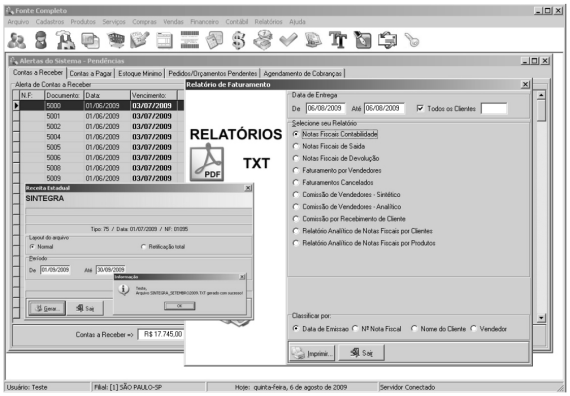
A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

DESENVOLVEDOR, QUANTO VALE O SEU TEMPO?

SISTEMA EMPRESARIAL COMPLETO E COM CÓDIGO-FONTE.



Consulte valores e condições em nosso site.

- Sistema projetado exclusivamente para empresas (todos os portes) que trabalham com comércio e prestação de serviços.
- Desenvolvido em linguagem Delphi 2007.
- Componentes incluídos e com todos os fontes.
- Banco de dados (script incluído) PostgreSQL 8.x ou Firebird 2.x.
- Ambiente client/server.
- Script de instalação (Inno Setup) incluído.

ESTRUTURA:
- Código-fonte documentado;
- Banco de dados documentado;
- Regras de negócio;
- Integridade referencial;

Basta somente compilar no Delphi. (somente Delphi 2007 ou superior)

www.fontecompleto.com.br | vendas@fontecompleto.com.br

Nesta seção você encontra artigos intermediários sobre Delphi Win 32 e Delphi for .NET

Solução para relatórios WEB

Integre sua aplicação com Web Service e crie relatórios com o Rave Reports

Maikel Marcelo Scheid
maikelscheid@gmail.com
É técnico em Informática com ênfase em Análise e Programação de Sistemas. Atua na área de Desenvolvimento de Softwares em Delphi para plataforma Win32 e .NET com banco de dados Firebird e MS SQL. É membro da equipe Editorial ClubeDelphi.

Resumo do DevMan

Comumente o uso de relatórios do Rave Reports ocorre apenas em aplicações Delphi Win32 devido a falta de suporte da ferramenta em aplicações Web. Neste artigo veremos uma solução que contorna o problema e possibilita que o mesmo projeto de relatórios seja utilizado no sistema Desktop e também pela aplicação integrada Web.

Nesse artigo veremos

- Criação de relatórios com conexão a dados nativa no Rave Reports;
- Criação de Web Service capaz de conectar-se a arquivos de relatórios do Rave Reports e a partir deste exportar relatórios para PDF;
- Criação de aplicação ASP.NET e consumo de Web Services para gerar relatórios e exibição do PDF no browser.

Qual a finalidade

- Criar relatórios a partir do Rave Reports para plataforma Web.

Quais situações utilizam esse recurso?

- Em aplicações web com ASP.NET onde há necessidade de exibir relatórios com tamanhos de página customizados para impressão.

A criação de relatórios para a plataforma Web acaba na maioria dos casos sendo uma preocupação a mais ao desenvolvedor na hora de criar e disponibilizar os recursos na sua aplicação Web. Por mais que haja ferramentas que possibilitem a criação destes relatórios para a Web, em muitos casos as habilidades de gerenciar a ferramenta ou o custo de uma licença de uso podem se tornar um empecilho de utilização da mesma, problema que conseguiremos contornar ao usar o Rave Reports Borland Edition, versão que acompanha a própria instalação do Delphi e que possui componentes de conexão nativa aos tipos de banco de dados mais utilizados.

Como sabemos, o Delphi for .NET não oferece compatibilidade nativa com projetos do Rave Reports e por isto, vamos criar uma aplicação intermediária que faça a conexão com o projeto de relatórios e disponibilize o resultado no formato PDF para que possa ser consumido pela aplicação .NET. Estamos falando da criação de um Web Service

que ao ser solicitado fará uma chamada ao projeto de relatório, configurando a conexão do mesmo a base de dados, envio de parâmetros para comandos SQL quando necessário e gerando um relatório em formato PDF que será salvo no disco do computador e devolvendo o caminho destino a aplicação Web que por sua vez fará apenas exibição de um PDF no browser, este então que poderá ser salvo ou impresso pelo usuário.

Para desenvolver toda esta estrutura, definiremos primeiramente como banco de dados o uso do exemplo Employee.fdb que acompanha a instalação do Firebird e em seguida passaremos ao Rave Reports onde criaremos um relatório para exibição de clientes de acordo com o país selecionado que passaremos em forma de parâmetro para a SQL configurada no Rave. O próximo passo será a criação e configuração de funções e procedimentos em um Web Service onde criaremos os componentes de comunicação ao Rave Reports em tempo de execução fazendo toda sua configuração, e por último, a criação de uma aplicação ASP.NET que a partir da listagem de países irá consumir e realizar as chamadas ao Web Service recebendo como resultado um caminho de onde foi salvo o PDF e exibindo o mesmo na tela ao usuário.

Criando o projeto do relatório – Rave Reports

A partir do Delphi aberto, vá ao menu Tools>Rave Reports Designer para abrir a ferramenta de criação de relatórios, e através do menu File>New crie um novo arquivo e salve o mesmo como "prjRelatorio.rav" em um diretório chamado "Reports" em "C:\Inetpub\wwwroot\". Na paleta ao lado direito da folha do relatório (Report Library), altere o nome do mesmo para "_Clientes" e defina o tamanho da página na propriedade Paper Size como "A4 Sheet, 210- by 297-millimeters" (**Figura 1**).

Não possuindo suporte ao uso de objetos DataSet para compartilhamento de dados entre a aplicação Delphi e o relatório, faremos o uso de conexões nativas do Rave e passaremos a utilizar instruções de SQL diretamente na ferramenta, onde obteremos os campos a serem exibidos no formulário. Para criar uma conexão, vá ao menu File>New Data Object e selecione o item Database Connection e avance para a próxima etapa onde precisaremos escolher o tipo de conexão compatível com o banco de dados a ser utilizado. Ao utilizar o banco de dados Firebird, escolha a opção IBX e finalize esta tela, onde seremos direcionados a configuração dos parâmetros da conexão, definindo assim um protocolo de conexão ("TCP/IP"), servidor do nosso banco de dados ("127.0.0.1"). Em Database File informaremos o caminho da base de dados ("C:\Arquivos de Programas\Firebird<versão>\examples\Employee.fdb") e no dialeto de SQL selecione o dialeto 3 ("Dialect 3"). Clicando na guia Authentication precisamos configurar o nome do usuário ("SYSDBA") e a senha ("masterkey"). Para testar se a configuração da conexão está correta, clica na aba Test e em seguida clique em Test Connection, listando assim todas as tabelas da base. Confirme a configuração clicando em Ok.

Observe que um novo objeto (Database1) que é a conexão com a base de dados foi criado no Data View Dictionary, e ao selecionar esta observe que possui duas propriedades para configuração da conexão com a base de dados, AuthDesing e AuthRun sendo uma delas a configuração dos parâmetros para utilização em tempo de design e outra para utilizar quando o projeto é executado.

Para ter acesso aos dados das tabelas precisamos adicionar um novo componente em File>New Data Object e selecione o item Driver Data View que funciona semelhante a um SQLDataSet quando comparado aos componentes da paleta dbExpress. Clique em Next e selecione em seguida o componente de conexão Database1 criado anteriormente e finalize esta etapa onde será exibido um Query Advanced Designer semelhante ao Query Builder do Ib Expert exibindo todas as tabelas da base para que seja montada a instrução SQL a ser aplicada. Arraste para a tela a tabela CUSTOMER onde por default já estará trazendo todos os campos da tabela. Para adicionar uma condição de parametrizar a busca de acordo com o país selecionado clique sobre o botão Editor onde teremos acesso á instrução SQL que está sendo gerada e adicionaremos de

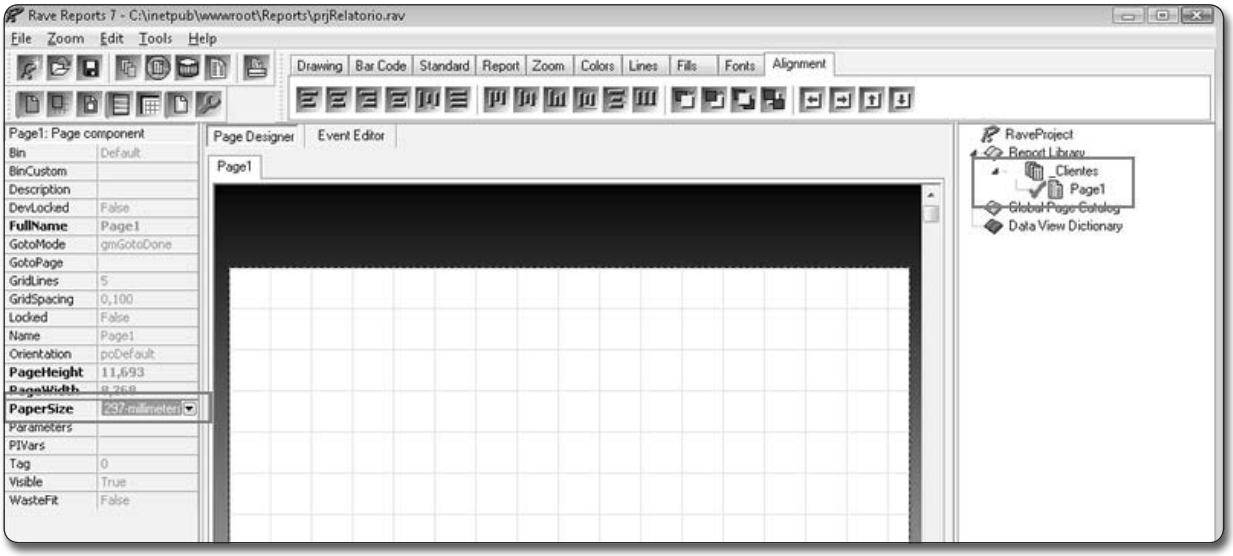


Figura 1. Configurando o nome e tamanho da página no Rave Reports

forma manual a condição Where deixando a SQL da seguinte forma e em seguida confirme a SQL e altere a propriedade Name para "dvCustomer":

```
Select T1.* From CUSTOMER T1
Where customer.country = ':country'
```

Ao clicar sobre o componente e abrir a listagem teremos acesso a todos os Fields resultantes da aplicação da instrução SQL. Um passo importante que cabe observar quanto a configuração de parâmetros em instruções SQL no Rave Reports é a utilização de aspas simples para identificação do parâmetro (':country'), o que faz com que o mesmo seja reconhecido e possibilite a passagem de valores para o preenchimento do mesmo. A passagem de valores ao parâmetro da instrução SQL será feita através de parâmetros que passaremos por meio do Delphi ao Rave Reports, onde no componente "dvCustomer", selecione a propriedade QueryParams de modo a abrir o editor e digite a seguinte instrução que irá atribuir valor ao parâmetro da SQL configurado (':country'), valor este que será recebido por meio de outro parâmetro que iremos criar em seguida e será recebido através da aplicação Delphi, o valor configurado entre parênteses significa a configuração de um valor Default caso o parâmetro não receba nenhum outro valor e pCountry nome que daremos ao parâmetro a ser criado na página:

```
country=Param.pCountry('USA')
```

Selecionando agora a pagina do relatório "_Clientes", clique sobre o item Page1 e vá até a propriedade Params abrindo o editor de configuração para que possamos adicionar o parâmetro "pCountry". Confirme e feche a tela após adicionado o parâmetro.

O próximo procedimento agora a ser realizado é a criação do layout do relatório, onde iremos adicionar os campos da SQL a serem exibidos na tela. Adicione na página da paleta Report um "Region Component" e dimensione-o de forma com que fique com uma pequena borda em ambas as extremidades da página. Logo em seguida arraste para dentro da região adicionada uma "Band Component" que será utilizada para adicionar o titulo do relatório e portanto, fixa no relatório e dentro desta arraste da paleta Standard um "Text Component" alterando a propriedade Text para "Relatório de Clientes", formate o titulo na propriedade Font e dimensione o tamanho do componente de acordo.

Para exibir os dados no relatório, arraste para logo abaixo da Band Component da paleta Report um "DataBand Component" e relacione sua propriedade DataView ao "dvCustomer". Com os Fields sendo exibidos no Data View Dictionary para o componente dvCustomer, utilize as teclas de atalho Ctrl para arrastar o DataText vinculado ao Field e a tecla Alt para arrastar o Text vinculado a informação a ser exibida, mantendo ambas as teclas pressionadas e utilizando o mouse para arrastar os objetos para tela, e arraste todos os componentes de modo que fiquem organizado de acordo com a ilustração da **Figura 2**, em seguida salve o relatório e pressione a tecla "F9" onde já poderá visualizar as informações sendo exibidas de acordo com o valor ("USA") atribuído ao parâmetro para a instrução SQL aplicada ao componente. Dessa forma concluídos a primeira etapa do processo onde criamos a estrutura do relatório por meio do gerador de relatórios Rave Reports.

Criando o Web Service para executar o relatório e criar o PDF

Criado e configurado o projeto de relatório no Rave Reports, veremos agora a criação de um Web Service que irá em runtime ao ser invocado criar os componentes necessários para a comunicação com o Rave, configurar o caminho do componente Database1 no projeto do relatório para conexão com a base de dados, realizar a configuração do parâmetro necessário para a busca de clientes de acordo com o estado, executar a pesquisa e salvar o relatório em formato PDF em um diretório que iremos receber por meio de parâmetro da aplicação asp.NET que iremos desenvolver no decorrer do artigo.

No RAD 2007 a ser utilizado para a criação do Web Service, no menu File>New>Other, escolha a categoria Delphi Projects>Web Services e selecione o item SOAP Server Application e confirme a tela. A tela seguinte irá questionar quanto ao tipo de serviços Web que iremos criar, onde mantenha selecionada a opção

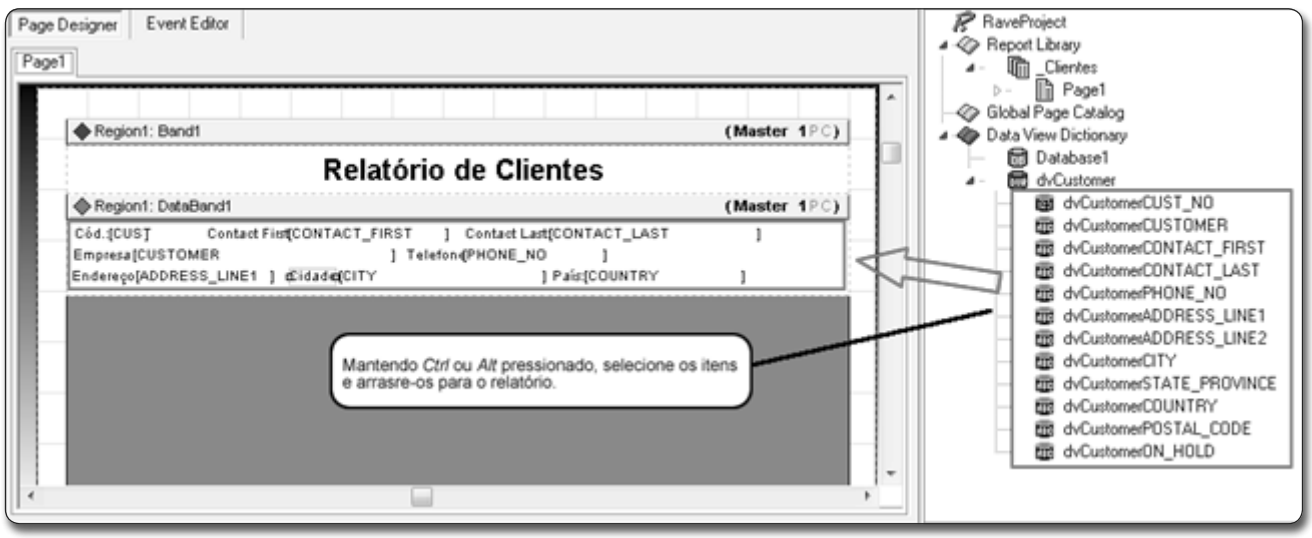


Figura 2. Adicionando os campos ao relatório

"ISAPI/NSAPI Dynamic Library" e confirmar a tela e em seguida confirme a mensagem para criação de uma interface para o serviço, onde deveremos informar os identificadores através de um Service Name ("rpRave") que será replicado em Unit indetifier para identificação dos arquivos a serem criados no projeto. Desmarque a opção para que não sejam gerados comentários e clique sobre o botão "Ok" na tela criando assim a estrutura do Web Service e toda a interface que utilizaremos para implementação das funções de criação do relatório e integração com outros serviços. Salve todos os arquivos da estrutura criada no mesmo diretório onde foi salvo o projeto de relatório criado no Rave Reports, deixando os nomes conforme sugeridos para o arquivo de interface ("rpRaveIntf.pas") e de implementação ("rpRaveImpl.pas") alterando o nome da unit principal para "urpReports.pas" e o projeto para "prjReports.dproj".

No Project Manager, abra o arquivo de interface ("rpRaveIntf.pas") onde iremos declarar a função a ser invocada pela aplicação asp.NET para a geração do arquivo PDF. Logo após ao código de registro da interface, declare a função conforme trecho de código seguinte, onde o nome da função será "ExecutaReport", recebendo como parâmetros de entrada o nome do relatório a ser executado, path para exportação do arquivo PDF, nome pelo qual o arquivo PDF será salvo e o valor do parâmetro á salvo e o valor do parâmetro country a ser repassado ao parâmetro da instrução SQL configurada no arquivo do relatório e como resultado obteremos um valor booleano identificando o sucesso ou o fracasso da tentativa de criação do PDF.

```
IrpRave = interface(IInvokable)
  ['{1E856281-37C0-4368-96DB-E7CA81DDAA96}']
    function ExecutaReport(NomeRelatorio,
PathPDF, NomePDF, pCountry:String; var retorno :
string):Boolean; stdcall;
```

Declarada a função na unit de interface, precisamos declarar de forma igual a função na unit de implementação ("rpRaveImpl.pas"), declarando-a na seção public criada e em seguida utilizando as teclas de atalho Shift + Ctrl + C para que o Delphi faça de forma automática a implementação do corpo da função onde codificaremos a rotina (**Listagem 1**).

Esta a única função que criaremos no Web Service, e é esta a função responsável por toda a parte de criação, relacionamento e configuração das opções e parâmetros de relatórios a serem exportados e exibidos no browser em formato PDF. Inicialmente definimos os objetos (componentes) que são responsáveis pela estrutura do relatório, objetos estes que são os mesmos adicionados em um Data Module de aplicações Win32. Em seguida, uma inicialização dos componentes é realizada e em seguida os mesmos são criados e suas propriedades configuradas em runtime.

O próximo bloco de código onde temos o camando With é o responsável por acessar o arquivo de relatórios, localizar o componente de conexão ("Database1") e definir os parâmetros para a conexão com a base de dados. No bloco seguinte, há as definições da tarefa a ser realizada com o relatório ao ser executado, onde definimos que ele será salvo em arquivo através da propriedade rdFile, definimos que o formato será para PDF.

Listagem 1. Rotina no WS para criação e configuração do relatório

```
function TrpRave.ExecutaReport(NomeRelatorio, PathPDF, NomePDF,
  pCountry: String; var retorno : string): Boolean; stdcall;
var
  Database: TRaveDatabase;
  RvProject1: TRvProject;
  RvSystem1: TRvSystem;
  RvRenderPDF1: TRvRenderPDF;
  SalvarArquivo : string;
  diretorio: string;
begin
  retorno := '';
  {Inicialização dos objetos}
  RvProject1   := nil;
  RvSystem1    := nil;
  RvRenderPDF1 := nil;
  try
    Result := False;
    try
      {Define uma instância para execução do relatório}
      RPDefine.DataID := IntToStr(HInstance);
      {Criação dos objetos em runtime}
      RvProject1   := TRvProject.Create(Application);
      RvSystem1    := TRvSystem.Create(Application);
      RvRenderPDF1 := TRvRenderPDF.Create(Application);
      {Configurando o projeto de relatório passando-lhe as configurações}
      RvProject1.Engine := RvSystem1;
      RvProject1.ProjectFile := gsAppPath{uses SWSystem}+'prjRelatorio.rav';
      {Abre o projeto e faz a configuração do caminho do banco de dados em
       tempo de execução. Estas informações você poderia armazenar em um
       arquivo de configurações *ini e ler em runtime}
      RvProject1.Open;
      With RvProject1.ProjMan Do
      Begin
        Database := FindRaveComponent('Database1',nil) as TRaveDatabase;
        Database.AuthRun.Datasource := '127.0.0.1:C:\Arquivos de
          programas\Firebird\Firebird_1_5\examples\EMPLOYEE.FDB';
        Database.AuthRun.Username   := 'SYSDBA';
        Database.AuthRun.Password   := 'masterkey';
      End;
      {Faz a configuração das opções, desabilita o preview do
       relatório, define que o relatório será exportado em formato
       PDF e seleciona o nome do relatório a ser executado}
      RvSystem1.SystemSetups := RvSystem1.SystemSetups -
       [ssAllowSetup];
      RvSystem1.DefaultDest := rdFile;
      RvSystem1.DoNativeOutput:= false;
      RvSystem1.RenderObject:= rvRenderPDF1;
      RvProject1.SelectReport(NomeRelatorio,True);
      {Passagem de parâmetros}
      RvProject1.SetParam('pCountry',pCountry);
      {Define os diretórios onde o PDF será exportado}
      diretorio:= PathPDF;
      if Copy(diretorio,Length(diretorio),1) <> '\' then
        diretorio := diretorio + '\';
      {Caminho e nome para o PDF}
      SalvarArquivo := diretorio+NomePDF+'.pdf';
      {Define o path do PDF para exportação}
      RvSystem1.OutputFileName:= SalvarArquivo;
      {Executa o relatório e faz com ele seja exportado}
      rvproject1.Execute;
      {Fecha o projeto do relatório e remove da memória}
      RvProject1.Close;
      RvProject1.FreeOnRelease;
      {Resulta True - êxito}
      Result := True;
      retorno := 'Relatório criado com sucesso!'
    finally
      RvProject1.Free;
      RvSystem1.Free;
      RvRenderPDF1.Free;
    end;
  except
    on e:exception do
    retorno := 'Ocorreu o seguinte erro:<br>'+e.Message;
  end;
end;
```

As próximas linhas são responsáveis pela passagem de valor aos parâmetros configurados no relatório e na instrução SQL, selecionamos o nome do relatório a ser executado e realizamos também a definição do diretório para onde o arquivo PDF será exportado, finalizando com a execução do relatório e atribuição de sucesso para o resultado da função, gerando assim o arquivo. Temos ainda um trecho com tratamento de exceções caso ocorram, a mensagem será exibida ao usuário.

Publicando a DLL Web Service no Windows XP + IIS 6.0

A publicação de Web Services com finalidade de criar relatórios não requer necessariamente que seja hospedada em um servidor com acesso a internet, sendo que a mesma poderá ser instalada de forma local, ou seja, apenas estará disponível para a aplicação que deverá consumi-la, mas nada impede que esta seja distribuída pela Web a fim de ser consumida por outras aplicações/plataformas. Nos ambientes de servidores com sistemas operacionais Windows XP, Windows 2000 e 2003 Server, a seqüência de publicação de uma DLL através do servidor Web IIS é a mesma de acordo procedimento que veremos adiante.

Nota

O IIS (Internet Information Services) não é instalado por padrão no Windows. Caso não tenha optado por ele durante a instalação do Windows, será necessário entrar no item Adicionar ou Remover Programas no Painel de Controle e efetuar a instalação.

Crie no servidor de hospedagem uma pasta (Reports) no diretório C:\Inetpub\wwwroot\ e copie para dentro do mesmo a DLL do Web Service (prjReports.dll) e o arquivo que contém os relatórios (prjRelatorio.rav), sendo necessário apenas estes arquivos para realizar a chamada do relatório e sua criação a partir da DLL. Para acessar o IIS, acesse o Painel de Controle>Ferramentas Administrativas>Gerenciador do Serviços de Informações da Internet (IIS). A sua esquerda aparecerá uma árvore de opções. Encontre a opção Sites da Web e dentro dela Default Web Site. Usaremos a opção de um novo diretório virtual possibilitando que você hospede a DLL em qualquer diretório do seu computador. Com o botão direito do mouse sobre a opção Default Web Site, selecione a opção Novo>Diretório virtual onde surgirá um Wizzard de configuração do diretório virtual. Ao avançar a primeira tela digite um apelido, ou seja, o alias Reports para o diretório e na tela seguinte selecione o diretório onde está hospedada a DLL C:\Inetpub\wwwroot\Reports\. Na tela de permissões de acesso, deixe selecionadas as opções Leitura, Executar scripts e Executar (aplicativos ISAPI ou CGI), sendo este último o responsável pela interpretação e execução da DLL (**Figura 3**). Realizado este processo das configurações confirme a tela e terá o diretório virtual criado. A partir do mesmo já poderá visualizar as funções disponíveis na DLL via browser. Para acessar o sistema via browser de agora em diante basta digitar o endereço a seguir:

```
http://localhost/reports/prjReports.dll
```

Ao executar o endereço do diretório virtual criado através do browser da Internet, uma página contendo as informações do serviço é exibida (**Figura 4**). Nesta página são publicados todos os métodos disponíveis na DLL, caminho de referencias e endereço do SOAP.

Figura 3. Permissões do IIS 6.0

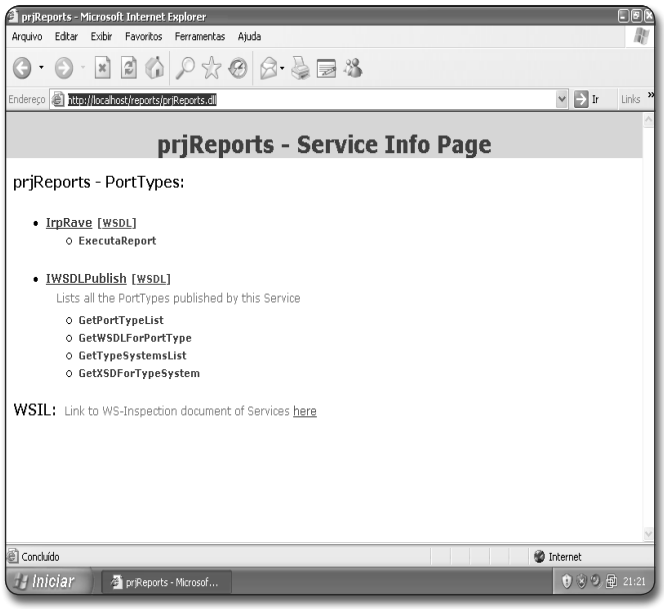


Figura 4. Página de informações do serviço

Publicando a DLL no Windows Vista + IIS 7.0

Os novos sistemas operacionais buscam sempre maneiras mais eficiente para controlar a segurança de ambientes mais inteligentes e com maior melhor interação com usuários. Uma destas ferramentas é o IIS 7.0 dos sistemas operacionais mais recentes, Windows Vista, Windows Seven e Windows Server 2008. Utilizando a DLL do Web Service criada, veremos a sua hospedagem da mesma agora em outro ambiente.

O caminho para acesso ao IIS 7.0, é o mesmo que vimos no tópico anterior. É importante ter previamente habilitado e instalado as funcionalidades de execução de scripts CGI e ISAPI. Iniciaremos com a criação de um diretório Reports em C:\Inetpub\wwwroot\, local para onde iremos copiar a DLL e o arquivo de relatórios criados. Após isso, utilizando o menu Iniciar>Executar digite inetmgr ou através do Painel de Controle abra o console de gerenciamento do IIS.

Semelhante ao processo do IIS 6, clique com o botão direito sobre Default Web Site e escolha a opção Adicionar Aplicativo (**Figura 5**), que trará um gerenciador. Digite o alias Reports, mantendo o Pool de Aplicativos como DefaultAppPool e definindo o caminho físico onde estamos hospedando o arquivo de DLL (C:\Inetpub\wwwroot\Reports) e confirme.

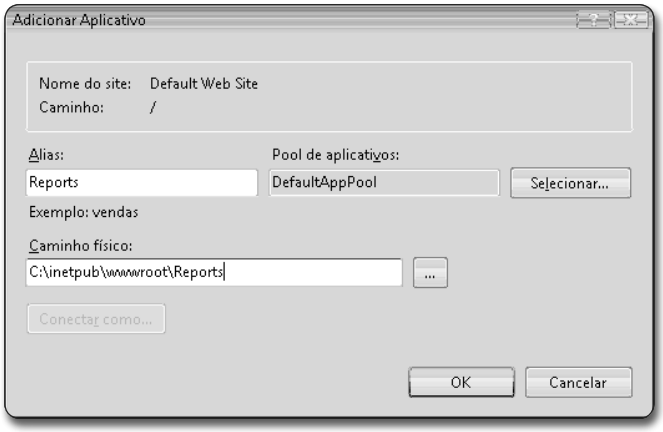


Figura 5. Adicionando novo aplicativo

Criado o aplicativo, precisaremos adicionar um novo mapa de script, ou seja, adicionar um novo manipulador Web funcionando como um mecanismo que ira responder por solicitações específicas, como ocorre no caso de DLL's e executáveis. Selecionando agora o aplicativo pedidos criado, observe ao centro do console de configurações do IIS um ícone chamado Mapeamentos do Manipulador. Acesse-o a fim de criar a modificação. Com o botão direito do mouse em qualquer parte, selecione a opção Adicionar Mapeamento de Script, preenchendo as informações de acordo com a **Figura 6**.

Conforme o próprio help do IIS nos mostra, em caminho da solicitação digite uma extensão de nome de arquivo com uma extensão para a qual o manipulador deve processar solicitações que será o nosso caso. O manipulador deverá executar todas as solicitações feitas para um arquivo específico, portanto digite o nome do arquivo e a extensão, por exemplo, prjReports.dll.

Para o executável, selecione a DLL no diretório que foi hospedada e atribua também um nome (Reports). Confirme as configurações da tela e confirme também a mensagem indicando a adição de um novo script. Ainda com o aplicativo Reports, selecionado na aba esquerda do console, selecione no rodapé do mesmo a opção Exibição de Conteúdo, listando assim a DLL relacionada.

Clique com o botão direito sobre a mesma escolhendo a opção Procurar, método que fará com que o browser padrão de internet seja executado carregando a página de informações do arquivo, concluindo assim a etapa de configuração.

Criando a aplicação Web para chamada e exibição do relatório

No menu File|New>ASP.NET Web Application – Delphi for .NET crie uma uma aplicação ASP.NET definindo seu nome como "WebReports" e mantendo o diretório "C:\Inetpub\wwwroot\Reports" e o servidor IIS como padrões. Esta nova aplicação será bem simples, nela teremos apenas um espaço para digitar o país para a busca conforme parâmetro configurado na SQL do Rave Reports e um botão para invocar a chamada ao Web Service e escrever o PDF na tela. Arraste então para o formulário um componente TextBox('txtbusca') e também um Button('btReport').

Antes de codificarmos o botão precisamos importar o caminho de referencia para o servidor do WebService. Localize

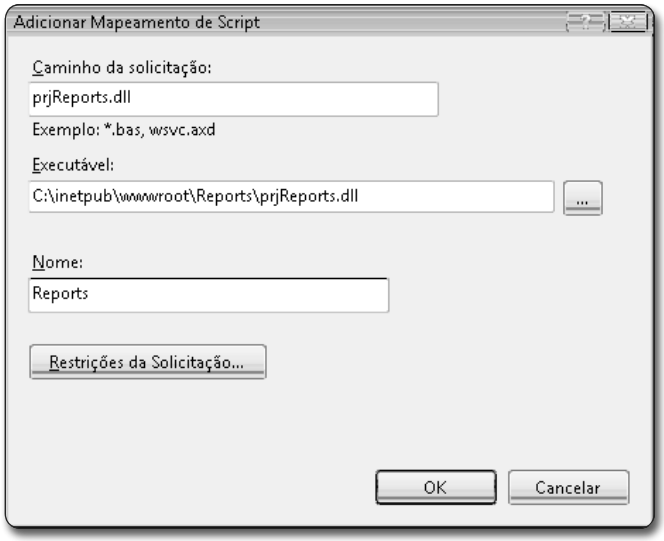


Figura 6. Adicionando mapeamento de script no IIS 7.0

no Project Manager a DLL da criação do seu projeto e com um clique do botão direito do mouse selecione a opção "Add Web Reference" que irá exibir um Wizzard para a adição do caminho ao arquivo principal do serviço. O caminho será obtido ao executarmos o WebService, e poderá ser copiado na URL do Webbrowser que por default será http://localhost/reports/prjReports.dll/wsdl/IrpRave. Digite o endereço na URL do Wizzard e utilize o botão GO ("1º") para chamar a página principal do serviço. Note que ao ser encontrada a página, o Wizzard habilitará automaticamente o botão Add Reference ("2º") que fará a importação da referencia (**Figura 7**). Ao importar o arquivo, observe que no Project Manager surgirá um novo item com as referencias externas adicionadas ao projeto (**Figura 8**).

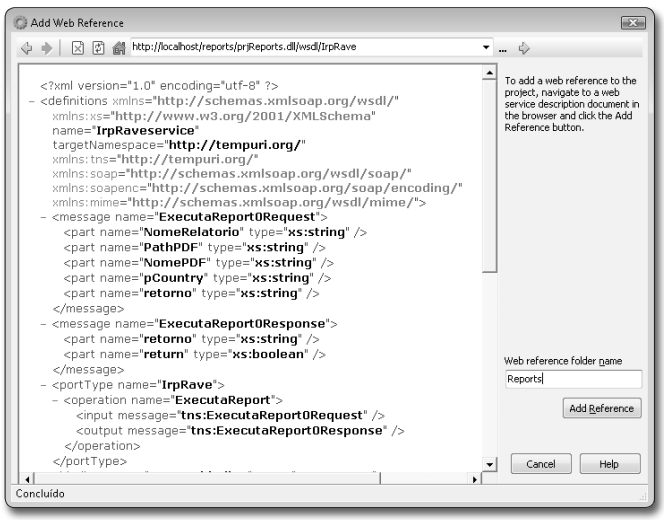


Figura 7. Adicionando a Web Reference

Adicionada a referencia ao Web Service, podemos agora realizar a chamada da função embutida no WebService adicionando o código da **Listagem 2** ao evento OnClick do botão btReport que se encontra comentado de acordo com a

ação de cada bloco de código. Adicione também referencia da página para a unit do WebService (Reports.IrpRaveservice) através do atalho Alt+F11;

O código adicionado ao evento OnClick do botão inicialmente recebe a definição de variáveis que serão criadas para instanciar objetos durante a execução bem como armazenar valores de localização e nome de arquivo. A criação de um objeto que define o relatório faz com que seja possível acessar sua função, e dessa forma, com a passagem de parâmetros que se possa realizar a chamada ao método responsável pela execução do relatório e criação do arquivo PDF que deverá ser exibido na tela. Após a chamada ao método, inicializamos um objeto de Stream que carrega para dentro da página o arquivo PDF gerado, que após realizada uma leitura, configuração e definição do objeto, o mesmo é escrito na tela (**Figura 9**). Execute a aplicação e digite o valor "USA" no campo de pesquisa e que será repassado à SQL configurada no Rave Reports. Faça a chamada do método e veja o relatório gerado no browser.

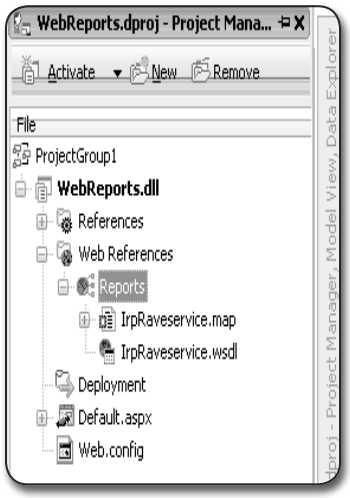


Figura 8. Referencias adicionadas ao projeto

Figura 9. Exibindo PDF criado na tela

Listagem 2. Codificando a chamada ao Web Service

```
procedure TDefault.btReport_Click(sender: TObject; e: System.
EventArgs);
var
  {declarando variáveis para criaçao do relatório}
  relatorio : Reports.IrpRaveservice.IrpRaveservice;
  retorno : string;
  path : string;
  fileName: string;
  Buffer: array of Byte;
  FileSize: Integer;
  strReport: string;
  {declarar uses System.IO}
  MyFileStream: FileStream;
begin
  {cria o objeto de referencia ao WS}
  relatorio := IrpRaveservice.Create;
  {Define diretório e nome do arquivo}
  path := 'C:\temp\';
  fileName  := 'rel_clientes';
  {faz a chamada ao WS. Aqui é criado o PDF}
  relatorio.ExecutaReport('_Clientes',path,fileName, txtbusca.
   Text,retorno);
  strReport := path + fileName + '.pdf';
  {Cria nova rotina de stream carregando o PDF}
  MyFileStream := FileStream.Create(strReport, FileMode.Open,
   FileAccess.Read);
   while pos('\', strReport) > 0 do
    Delete(strReport, 1, pos('\', strReport));
  {faz a leitura do tamanho do PDF}
  FileSize := MyFileStream.Length;
  SetLength(Buffer, FileSize);
  MyFileStream.Read(Buffer, 0, FileSize - 1);
  MyFileStream.Close;
  Response.BufferOutput := True;
  Response.ClearContent;
  Response.ClearHeaders;
  {cria um novo cabeçalho na página do tipo PDF}
  Response.AddHeader('Content-Length', FileSize.ToString);
  {escreve o arquivo PDF no browser}
  Response.AddHeader('content-disposition', 'inline;filename=' +
   strReport);
  Response.ContentType := 'application/pdf';
  Response.BinaryWrite(Buffer);
  Response.Flush;
  Response.Clear;
end;
```

Conclusão

Vimos no decorrer deste artigo uma "nova" maneira de utilizar dos recursos do Rave Reports para criação de relatórios e exibi-los na Web. Por mais que sejam três aplicações distintas, em muitos casos a utilização desta rotina acaba trazendo informações e exibindo o resultado em um intervalo de tempo menor do que os métodos mais conhecidos, além de possibilitar a vantagem de que o Web Service seja hospedado em outro servidor garantindo performance para os aplicativos de forma individual. A partir desta estrutura criada, você poderá implementar no Web Services funções que deixem o mesmo dinâmico e pronto para receber qualquer número de parâmetros ou diferentes bases de dados, podendo hospedá-lo uma única vez e utilizar para diversas aplicações. Abraço e até o próximo artigo. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos para iniciantes na linguagem Delphi

Introdução à POO - Parte 2

Aprenda, de uma forma simples, a programar da forma correta

Guinther Pauli
guinther@devmedia.com.br
Tem 21 anos de experiência na área de desenvolvimento e arquitetura de software, dos quais 14 dedicados ao Delphi e 9 ao .NET. Certificado Borland em Delphi 6, 7, 2005, 2006, Delphi para Web e Kylix, Certificado Microsoft MCP, MCAD, MCSD. NET, MCTS e MCPD. Editor geral da Revista .NET Magazine Brasil e professor de Pós-Graduação. Publicou mais de 150 artigos, 400 vídeo-aulas, palestrante em 5 edições da Borland Conference. Especialista em desenvolvimento Delphi, C#, .NET, ASP.NET, atuando também em projetos de migração orientando empresas e desenvolvedores na adotação de melhores práticas.

Resumo do DevMan

Esta sem dúvida é a principal parte do nosso curso. Ensinar polimorfismo não é algo fácil. Implementá-lo já é complicado, entendê-lo é pior ainda, saber como usá-lo em uma situação real, quase impossível (mas nós vamos ver como, no final do curso!). Herança e polimorfismo, duas peças-chave da orientação a objetos, são os principais temas desta segunda parte da série. Você verá também o que são métodos estáticos e virtuais. Como compartilhar comportamento, como herdar características, como herdar funcionalidade, como anular ou adicionar funcionalidade, como abstrair a complexidade usando herança e classes básicas. Entenderá também porque muitas coisas da VCL foram implementadas da forma como vemos hoje, graças a comparativos feitos ao longo do texto.

Nesse artigo veremos

Este artigo aborda a orientação a objetos com o Delphi, usando uma metodologia simples, didática, de fácil aprendizado. Veremos na teoria, e também na prática, todos os conceitos, fundamentos e recursos oferecidos pelo Delphi para promover a POO.

Qual a finalidade

A POO pode e deve ser aplicada de forma inteligente, ela serve para construir sistemas mais robustos, mais confiáveis, de fácil manutenção, que permitam maior reaproveitamento de código.

Quais situações utilizam esse recurso?

A POO é útil em qualquer sistema, seja ele Web, Desktop, Mobile, não importa o tipo de aplicação. Os conceitos aqui apresentados, em um exemplo simples, podem ser utilizados em aplicações reais, como é apresentado em um estudo de caso no final desta série.

No artigo da edição anterior, aprendemos importantes conceitos da orientação a objetos, como Abstração, Encapsulamento, Herança e Polimorfismo. Vimos tudo isso na teoria, por enquanto, pois conforme eu mesmo comentei, a OO é bastante conceitual. E para podermos aplicá-la na prática, precisamos ter esses pilares bem fundamentados.

Vamos continuar nos aprofundando na orientação a objetos com o Delphi, desta vez, entendo o que considero os principais recursos da OO que são aplicados no Delphi: herança e polimorfismo. A

herança, na Delphi Language, é conseguida através do uso da palavra-chave class, antigamente object (que por questões óbvias mudou, visto que uma classe não é um objeto). O polimorfismo é conseguido aplicando-se várias palavras-reservadas, como virtual, dynamic, abstract e override. Veremos tudo na prática em nosso exemplo.

Iniciamos nossos trabalhos criando uma interface que permitia manipular carros e aviões, através de Edits. O usuário informava valores para as propriedades dos objetos, que podiam ser do tipo TCarro e TAviao, e então criava instâncias dessas classes. Aqui já temos uma boa prática sendo aplicada, nossas classes de negócio estão separadas do código de interface (o formulário). Vamos então conhecer o que é a herança e como ela funciona.

Herança

Nosso exemplo da edição anterior terminou com um grave defeito de modelagem. Claro, proposital. Vamos observar o código das classes TCarro e TAviao (**Listagem 1**).

Como você deve ter notado, muitas características estão presentes tanto em TCarro com em TAviao, como os campos Descricao, Capacidade e o método Mover. O que faremos agora é generalizar as classes TCarro e TAviao, criando uma classe base chamada TMeioTransporte, que conterá as características comuns a todos os meios de transporte. Dessa forma, TAviao e TCarro herdarão dessa classe e adicionarão funcionalidades específicas.

Listagem 1. TCarro e TAviao

```
unit uCarro;

interface

type
  TCarro = class
    Descricao : string;
    Capacidade : integer;
    Quilometragem : integer;
    procedure Mover();
  end;

implementation

uses Dialogs;

{ TCarro }

procedure TCarro.Mover();
begin
  ShowMessage(Descricao+' entrou em movimento.');
end;

end.
-------------------------------------------------
unit uAviao;

interface

type
  TAviao = class
    Descricao : string;
    Capacidade : integer;
    HorasVoo : integer;
    procedure Mover();
  end;

implementation

uses Dialogs;

{ TAviao }

procedure TAviao.Mover();
begin
  ShowMessage(Descricao+' está voando.');
end;

end.
```

Esse é um dos princípios mais fundamentais para permitir a reutilização de código. Antigamente, na programação estruturada, conseguíamos reutilizar código criando units com rotinas (procedures e functions) que fossem bastante parametrizadas, sem depender uma da outra, que pudessem funcionar bem separadamente. Na orientação a objetos, a herança é muito, mas muito mais superior do que isso, como vamos comprovar.

Clique em File>New>Unit. Salve a nova unit como "uMeioTransporte.pas". Defina a nova classe como mostrado a seguir na **Listagem 2**. Nossa nova classe TMeioTransporte define o comportamento básico de todos os meios de transporte, sejam eles aviões ou carros (ou outros que venhamos a criar). Todo meio de transporte tem uma descrição e uma capacidade, que são atributos. Todos possuem a capacidade de se movimentar, que é sua funcionalidade. Então, podemos dizer que uma classe é basicamente a união de dados mais funcionalidade.

Nota do DevMan

Quem desenvolveu em Pascal antigo, deve lembrar que não era possível criar estruturas que contivessem dados e funcionalidades (somente a partir do Turbo Pascal 5.5), era necessário criar um record (o struct do C) e colocar a funcionalidade que trabalhava essas informações em código à parte.

Listagem 2. Código da classe TMeioTransporte

```
unit uMeioTransporte;

interface

type
  TMeioTransporte = class
    Descricao : string;
    Capacidade : integer;
    procedure Mover();
  end;

implementation

{ TMeioTransporte }

procedure TMeioTransporte.Mover();
begin

end;

end.
```

Ok, já definimos nossa classe base. Agora vamos fazer TCarro herdar de TMeioTransporte, ou seja, todas as características definidas na classe base passarão a ser acessíveis na classe descendente (desde que sejam públicas ou protegidas – veremos mais sobre especificadores de visibilidade futuramente). Abra então a unit uCarro.pas e altere a definição da classe TCarro como mostrado na **Listagem 3**.

Os campos Capacidade e Descricao são agora herdados de TMeioTransporte. Mas e o método Mover, por que não foi removido? Porque nem todos os meios de transporte se movimentam da mesma forma, ou seja, cada descendente deve implementar da sua forma o método Mover. O que fizemos foi apenas declarar o método Mover na classe base para dizer que ele existe para todas as classes descendentes. Falaremos

Listagem 3. Classe TCarro agora herda de TMeioTransporte

```
unit uCarro;

interface

uses
  // coloque essa unit p/ acessar a classe TMeioTransporte
  uMeioTransporte;

type
    // observe que TCarro agora herda de TMeioTransporte
  TCarro = class(TMeioTransporte)
    // observe que retiramos os campos Capacidade e Descricao daqui
    Quilometragem : integer;
    procedure Mover();
  end;

implementation

uses Dialogs;
{ TCarro }

procedure TCarro.Mover();
begin
  ShowMessage(Descricao+' entrou em movimento.');
end;

end.
```

mais sobre isso a seguir, quando estudarmos o polimorfismo.

Repita os mesmos passos para a classe TAviao, fazendo a herança e removendo os campos Capacidade e Descricao. O novo código da classe TAviao é mostrado na **Listagem 4**.

Feito. Simples, fácil, direto, produtivo, elegante e prático. E funcional.

Listagem 4. Classe TAviao agora hera de TMeioTransporte

```
unit uAviao;

interface

uses
  // coloque essa unit p/ acessar a classe TMeioTransporte
  uMeioTransporte;

type
// observe que TAviao agora herda de TMeioTransporte
  TAviao = class(TMeioTransporte)
// observe que retiramos os campos Capacidade e Descricao daqui
    HorasVoo : integer;
    procedure Mover();
  end;

implementation

uses Dialogs;

{ TAviao }

procedure TAviao.Mover();
begin
  ShowMessage(Descricao+' está voando.');
end;

end.
```

Métodos Estáticos

Existem dois tipos de métodos no Delphi: estáticos ou virtuais. Por padrão, todos os métodos de uma classe são estáticos, a menos que você indique que são virtuais através das palavras-chave virtual ou dynamic. Não se preocupe, vamos conhecer como funcionam essas palavras logo a seguir. Diferente de outras linguagens, não existe a necessidade de se usar algo como static para indicar que o método é estático.

Mas qual é a diferença entre um método estático e virtual. Essa é a peça-chave para o polimorfismo, e aqui, ao invés de fazer analogia com o mundo real para entendermos esse

Nota do DevMan

A palabra-reservada dynamic surgiu no Delphi 4, é a mesma coisa que virtual, porém instrui o compilador a gerar uma VMT (Virtual Method Table) mais otimizada, especialmente para cadeias de classes com grande hierarquia (muitos ancestrais).

funcionamento, vamos direto ao ponto, na prática. Vá até o formulário principal e localize a seguinte declaração, na seção public da classe:

```
public
  Carro : TCarro;
  Aviao : TAviao;
```

Substitua por:

```
public
  Carro,Aviao : TMeioTransporte;
```

Declare uMeioTransporte na cláusula uses. Se você compilar o programa agora, receberá erros. Como a variável Carro agora é um TMeioTransporte ela não reconhece mais o campo Quilometragem. Faça então o seguinte TypeCasting:

```
TCarro(Carro).Quilometragem:=StrToIntDef(EdtQuilom
etragem.Text,0);
```

Faça o mesmo para a atribuição de HorasVoo de TAviao:

```
TAviao(Aviao).HorasVoo:=StrToIntDef(EdtHorasVoo.Text,0);
```

Isso que acabamos de fazer (RTTI) é uma má prática, má prática mesmo. Vou explicar mais adiante o porquê, quando estudarmos o polimorfismo, e assim resolveremos o problema de uma forma muito mais elegante.

Execute a aplicação. Você verá que tudo funciona perfeitamente, menos o botão Mover. Quando você chama o método Mover, você está chamado o método da classe TMeioTransporte, que não possui implementação. Isso porque Mover é um método estático (o padrão), e dessa forma ele fica vinculado à classe que você *declarou*. Observe a declaração das variáveis Aviao e Carro:

Nota do DevMan

TypeCasting é um recurso da linguagem que permite converter um tipo de objeto em outro, desde que sejam compatíveis. O typecast pode ser feito de duas formas em Delphi:

Estilo C, direto:
TTipoDesejado(Objeto)
Com operador RTTI "as":
(Objeto as TTipoDesejado)

A diferença entre o primeiro e o segundo é que, no estilo C, o compilador não usa RTTI (Runtime type information) para verificar se você está realmente convertendo um objeto compatível. Com o "as", o compilador nos bastidores faz um teste como outro operador RTTI, o "is", é mais seguro, mas mais lento (RTTI é sempre lento). Então, prefira usar o estilo C desde que tenha certeza do que está convertendo.

```
public
  Carro,Aviao : TMeioTransporte;
```

Observe agora como foram criados Carro e Aviao:

```
Aviao:=TAviao.Create;
Carro:=TCarro.Create;
```

Como você viu é possível declarar uma variável de um tipo e instanciar a partir de outro tipo (desde que seja descendente). Isso é possível porque TAviao e TCarro são TMeioTransporte. A recíproca não é verdadeira. Nem todo TMeioTransporte é um TAviao ou TCarro. Outro exemplo, na vida real, podemos dizer que gatos e cachorros são animais. Mas nem todo animal é um gato ou cachorro.

Quando você chama o método Carro.Mover, não importa se a instância atual faz referência a um TCarro. Como o objeto foi declarado com TMeioTransporte, o método Mover sempre será dessa classe (pois, novamente, ele é estático).

Veja que a Delphi Language, assim como o antigo Pascal e muitas outras linguagens de programação, é uma linguagem fortemente tipada. Não podemos, por exemplo, colocar uma string dentro de uma variável inteira. Porém, se usarmos herança, podemos atribuir objetos a diferentes tipos desde que tenha um ancestral comum.

Isso é amplamente utilizado na VCL do Delphi. Você já parou para se perguntar por que o gerador de relatórios Quick Report aponta para um TDataSet e não um TClientDataSet? Por que os controles data-aware são ligados a um TDataSource, que por sua vez se liga também a um TDataSet?

TDataSet funciona como um "contrato". Ele foi definido há mais de 10 anos, como poderiam os engenheiros do Delphi adivinhar que tecnologias de acesso a dados seriam criadas nos anos seguintes? BDE, ADO, dbExpress, Zeos, ADO, ClientDataSet etc., todas jogam as regras de TDataSet, ou seja, implementam seus métodos virtuais, implementam o comportamento padrão. Logo, qualquer objeto que saiba trabalhar com esse padrão, poderá usar seus recursos sem conhecer detalhes sob sua implementação, isso é abstração.

É por isso que o Quick Report funciona (pelo menos em teoria) com todos esses engines de acesso a dados. E por isso que os controles data-aware independem de tecnologia de acesso. TDataSet esconde, abstrai, serve como base, um protocolo, um padrão. Não fosse isso, teríamos que ter classes especializadas para tratar cada uma das tecnologias de acesso, algo como TQuickReportForBDE, TQuickReportForDBX, TQuickReportForADO, ou pior ainda, TDBEditForBDE, TDBEditForDBX, TDBEditForClientDataSet (imagine multiplicar todos os tipos de TDataSets por todos os tipos de controles data-aware, certamente o IDE do Delphi precisaria de uma centena de paleta de componentes e o seu computador alguns GB de memória a mais).

Não é assim que funciona na vida real, não é assim que funciona na OO, não é assim que funciona na Delphi Language. Imagine que se para cada tipo de impressora, você tivesse que ter uma porta diferente do micro? Algo como PortaHP, PortaCanon, PortaElgin (agora forcei, lembrei da minha Lady 90). E se criarem um novo tipo, você não vai querer ir na fábrica do seu note para abrir um buraco na lateral e conectar uma impressora nova? Não é assim que as coisas funcionam. Inventaram um protocolo, um padrão, uma interface (veremos que interfaces na OO têm exatamente esta função), por exemplo a USB, e pronto, você conecta qualquer dispositivo ao seu micro, desde que siga esse padrão. A mesma coisa com o TQuickReport, com os controles data-aware, e por aí vai.

Lembra quando eu falei que usar typecast era horrível? Pois bem, agora você entenderá o porquê. Se não fosse o polimorfismo, uma classe que precisasse se ligar a outra teria que testar um por um dos possíveis tipos para poder chamar um método, como na **Listagem 5**. E o pior, se um novo tipo fosse criado (o que seria a coisa mais comum do mundo), precisaríamos abrir a classe e incluir novo código. Na mesma listagem, você vê a diferença brutal ao se optar por polimorfismo (nunca prefira RTTI ao polimorfismo, nem se precisar recorrer à POG).

Tudo bem, já entendemos que isso só funciona com polimorfismo, não adianta apenas abstrair. As coisas mudam constantemente na vida real. Na tecnologia, pior ainda. Então precisamos preparar nosso código para as mudanças, que venham a acontecer. Mudar, tomar outra forma, uma nova implementação (desde que siga o padrão), isso é polimorfismo.

Métodos Virtuais e Dinâmicos - Polimorfismo

Vamos então ver como isso funciona na prática em nosso exemplo. Abra a unit uMeioTransporte e altere a declaração do método Mover como mostrado a seguir:

```
procedure Mover; virtual;
```

Virtual define um método que pode ser sobrescrito e assim tomar múltiplas formas em classes descendentes. Sobrescrito significa anular a implementação da classe base, fornecer uma outra forma de implementação, uma forma diferente. No entanto,

Nota do DevMan

A partir do Delphi 7, a linguagem Object Pascal utilizada pelo Delphi, por questões de marketing, passou a se chamar Delphi Language. Então quando você disser "Delphi é uma linguagem" e alguém responder, "Não, Delphi é a ferramenta, a linguagem é o Pascal", saiba que você está certo.

Listagem 5. Type Casting/RTTI vs. Polimorfismo

```
procedure TFormulario.MoverMeioTransporteDeFormaBurra();
begin
   // sem polimorfismo
  if MeioTransporte is TCarro then
              (MeioTransporte as TCarro).Mover();
  if MeioTransporte is TAviao then
              (MeioTransporte as TAviao).Mover();
  if MeioTransporte is TBicicleta then
              (MeioTransporte as TBicicleta).Mover();
  if MeioTransporte is TBarco then
              (MeioTransporte as TBarco).Mover();
  if MeioTransporte is TTrem then
              (MeioTransporte as TTrem).Mover();
  if MeioTransporte is TCarroca then
              (MeioTransporte as TCarroca).Mover();
  if MeioTransporte is TDromedario then
              (MeioTransporte as TDromedario).Mover();
  if MeioTransporte is TOvni then
              (MeioTransporte as TOvni).Mover();
  if MeioTransporte is TMula then
              (MeioTransporte as TMula).Mover();
        ...
end;

procedure TFormulario.MoverMeioTransporteDeFormaInteligente();
begin
   // com polimorfismo
       MeioTransporte.Mover();
end;
```

podemos sobrescrever para adicionar mais funcionalidade, nesse caso, herdando funcionalidade da classe base e incluindo mais comportamento. Sim, isso é possível, com o uso de inherited.

Como se sobrescreve um método virtual? Indo direto ao ponto, com override. Abra a unit uCarro.pas e altere o método Mover como mostrado a seguir:

```
procedure Mover(); override;
```

Faça o mesmo para TAviao.Mover().

Dessa forma, se você agora chamar o método Mover de um objeto instanciado como TCarro, mas originalmente declarado como TMeioTransporte, fará uma chamada ao método TCarro.Mover, pois o método deixou de ser estático e passou a ser virtual (**Figura 1**), ou ainda conhecido como método polimórfico.

Figura 1. Chamando um método virtual

Inherited

Como você pode ver, o método Mover não possui implementação na classe base TMeioTransporte. Podemos implementar o comportamento padrão nessa classe e deixar que as classes descendentes chamem esse método, adicionando alguma funcionalidade mais específica.

Imagine que, por exemplo, todo o veículo precisa ser ligado antes de entrar em movimento. Você não teria que implementar isso em todas as classes descendentes de TMeioTransporte, pois esse comportamento é padrão para todas. Abra então a unit uMeioTransporte.pas e implemente o método Mover como mostrado na **Listagem 6**.

Declare Dialogs na cláusula uses. Agora localize a classe TCarro, vá até a implementação do método Mover e adicione a diretiva inherited como mostrado na **Listagem 7**.

Execute a aplicação. Adicione um carro à lista e clique no botão Mover. Como o método é virtual, será disparado o método Mover de TCarro. A primeira linha da implementação desse método contém um inherited, o que faz uma chamada ao método Mover da classe base (TMeioTransporte), e logo em seguida é executado o restante do código. Por último, chame inherited também no método Mover de TAviao. Veja o resultado na **Figura 2**.

Listagem 6. Implentando código na classe base

```
procedure TMeioTransporte.Mover;
begin
  ShowMessage('Ligando '+Descricao);
end;
```

Listagem 7. Mover chama método de mesmo nome na classe base

```
procedure TCarro.Mover;
begin
  inherited;
  ShowMessage(Descricao+' entrou em movimento');
end;
```

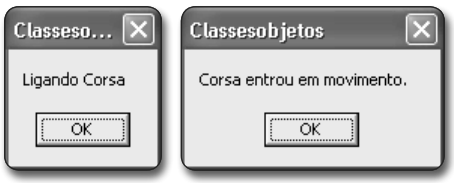


Figura 2. Inherited chama o método da classe base

Esse recurso é bastante poderoso, visto que podemos compartilhar uma mesma implementação herdada da classe base e adicionar alguma funcionalidade específica. Em uma linguagem não orientada a objetos, você provavelmente precisaria criar procedimentos diferentes e fazer vários testes condicionais com IF para saber que método deve ser chamado (como na **Listagem 5**).

Sugestão de leitura

Se quiser se aprofundar um pouco mais no lado "obscuro" do Delphi, sugiro a leitura da obra "Delphi: O Guia Essencial", do autor Ray Lischner, editora O'Reilly. Das mais de 50 obras de Delphi da minha coleção, é a que mais detalha aspectos avançados da linguagem e VCL.

Conclusão

Sem dúvida, há muito mais a se falar sobre herança e polimorfismo. E sem dúvida, vamos ver muito mais sobre essas duas técnicas ainda neste curso. Mas com certeza você já viu que o recurso não está disponível por opção na linguagem, é obrigatório utilizá-lo. Sei que muitas dúvidas pairam a sua mente agora: se isso é realmente fantástico, como podemos aplicar o que aprendemos em meu sistema real, client/server com Firebird, dbExpress e ClientDataSet, que usa formulários, DataModules e controles data-aware?

Que tal começar mudando não o seu código, mas os seus conceitos, a sua mente, a sua maneira de ver as coisas? O meu objetivo aqui foi mostrar para você amigo leitor, como usar o polimorfismo e herança de uma forma simples, em um exemplo simples, com objetos e classes mais simples ainda. Mas não se preocupe nem duvide, no final, mostrarei um exemplo realíssimo de como utilizar essas técnicas para obter produtividade, reaproveitamento de código, maior facilidade na manutenção, boas práticas e finalmente, um código bem mais profissional.

Temos muitas coisas interessantes ainda pela frente neste pequeno exemplo! Nas próximas partes, veremos métodos abstratos, construtores e destrutores, especificadores de visibilidade, propriedades, métodos get / set e finalmente, vamos dar um passo além transformando nossas classes em componentes, e ligando meios de transporte a pessoas (da mesma forma que o QR se liga a um TDataSet), consolidando ainda mais os conceitos de abstração, herança e polimorfismo.

Grande abraço e até a próxima edição. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Nesta seção você encontra artigos sobre a linguagem PHP e a ferramenta Delphi for PHP

Trabalhando com XML no PHP

Aprenda a mapinular estes tipos de arquivos com classes nativas do PHP

Rodrigo Carreiro Mourão

rodrigocarreiro@rmfactory.com.br

Apaixonado por tecnologia da informação. Atualmente é sócio diretor da RM Factory Soluções em TI, empresa especializada em consultoria na área de TI e desenvolvimento de softwares customizados, parceira da TDS Tecnologia - RJ. Possui os títulos de BDS2006 Win32 Product Certified e Borland Instructor Certified. Como instrutor de treinamentos oficiais já ministrou treinamentos para centenas profissionais e servidores de órgãos como TJ-RJ, JF, BB, IBGE, MB, EB, etc. Palestrante da Borland Conference 2007. Mantenedor da comunidade www.delphi-soPHP.com, a primeira do Brasil 100% feita com Delphi for PHP. Autor de diversos artigos técnicos veiculados nos principais sites e revistas sobre Delphi do Brasil .

Resumo do DevMan

Já vimos em diversos artigos publicados aqui mesmo na revista ClubeDelphi que uma das palavras mais utilizadas em Tecnologia da Informação hoje em dia é: integração. Integrar sistemas não é mais nenhum diferencial de software, é quase uma obrigação. No passado uma das formas mais utilizadas para integrar diferentes sistemas em diferentes plataformas era o arquivo TEXTO.

Hoje precisamos de soluções mais robustas e simples de implementar. É por esse motivo e, claro por diversos outros, que o formato XML surgiu. Nesse artigo veremos um pouco da sua história, como e onde implementar e ainda como desenvolver leitura e escrita desse tipo de arquivo em PHP.

Quem nunca ouviu falar, ou ainda, quem nunca precisou trabalhar com arquivos XML? Este tipo de arquivo se tornou muito popular devido a facilidade de se trabalhar com eles e a possibilidade de poder integrar-se com sistemas de diversas plataformas diferentes. Por ser auto descritivo, o XML facilita sua manipulação permitindo que se criem

Nesse artigo veremos

O artigo trata da classe de manipulação de arquivos XML no PHP utilizando DOM. Através desta classe iremos gerar e consumir arquivos XML de forma transparente sem se preocupar com detalhes da implementação do arquivo.

Qual a finalidade

Arquivos XML servem entre outras coisas para troca de informações em integrações B2B e B2C. Ele é muito utilizado também na confecção de WebService e criação e manipulação de arquivos de configuração. Por se tratar de um padrão mantido pela W3C temos ferramentas de validação que nos ajudam a verificar a integridade do arquivo.

Quais situações utilizam esse recurso?

Imagine uma empresa que possua sua matriz e diversas filiais e que por algum motivo, seja tecnológico ou não, não pode manter suas filiais on line full time. Assim surge a necessidade de integrar as informações das diversas filiais com a matriz. Neste contexto os arquivos XML auxiliariam bastante. Outra situação é quando tivermos a necessidade de armazenar dados de forma estruturada e que não irão sofrer acesso recorrente, para posterior consulta.

mecanismos de parser eficientes poupando o enfadonho trabalho que temos quando trabalhamos com manipulação de arquivos TXT.

Devido a estes e outros motivos o padrão XML é utilizado por diversas tecnologias existentes no mercado, como por exemplo os WebService. É muito comum vermos o uso de arquivos XML também em arquivos de configuração de aplicações como é o caso do Webconfig presente em aplicação ASP.NET. O parser de um arquivo XML é feito de forma extremamente rápida o que o levou a fazer parte de estruturas complexas de dados como é o caso dos Objetos DataSet presente no Framework .Net.

A Web é o ambiente em que vemos o padrão XML se aplicando em larga escala, por este motivo neste artigo veremos como trabalhar com este tipo de arquivo no PHP. Conheceremos as classes nativas da linguagem, seus principais métodos e recursos e como tirar o melhor proveito deles. Eu poderia ficar o artigo todo elencando as diversas tecnologias que lançam mão de arquivos XML em sua composição, mas ao invés disto vamos ver como manipulá-los na prática.

Resumo do DevMan

Business to Business - B2B é o nome dado ao comércio eletrônico associado a operações de compra e venda, de informações, de produtos e de serviços através da Internet ou através da utilização de redes privadas compartilhadas entre parceiros de negócios, substituindo assim os processos físicos que envolvem as transações comerciais. O B2B pode também ser definido como troca de mensagens estruturadas com outros parceiros comerciais a partir de redes privadas ou da Internet, para criar e transformar assim as suas relações de negócios.

Business to Customer/Client - B2C tem o mesmo sentido de B2B, porém na outra ponta estará a figura do cliente final, aquele que consome nossos produtos ou serviços.

Resumo do DevMan

XML (eXtensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um subtipo de SGML (acrônimo de Standard Generalized Markup Language, ou Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet. Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), RDF, SDMX, SMIL, MathML. (formato para expressões matemáticas), NCL, XBRL, XSIL e SVG (formato gráfico vetorial).

Um breve histórico sobre o XML

A origem do XML, ou melhor, de seu pai se é que podemos chamar assim data da década de 60 quando a IBM desenvolveu uma linguagem de marcação de documentos chamada GML. Este nome se deve ao fato de na época ter sido desenvolvida por Charles Goldfarg, Edward Mosher e Raymond Lorie. Anos mais tarde surge a SGML (Standard Generalized Markup Language) que veio a ser uma descendente da GML.

A SGML foi inicialmente concebida para permitir a partilha de documentos e sua leitura por máquinas em projetos de grande dimensão governamentais e na indústria aeroespacial. Estes documentos necessitavam permanecer legíveis por várias décadas. Por vota de 1993, o World Wide Web Consortium (W3C) começou a trabalhar em uma linguagem de marcação que combinasse a flexibilidade da SGML com a simplicidade da HTML. A idéia principal era poder criar uma linguagem que pudesse interagir com diversos softwares e que pudesse ser integrada a diversas linguagens.

Alguns pontos deveriam fazer partes desta nova linguagem como, por exemplo: possibilidade de criação de tags sem limite, interligação de bancos de dados distintos, separação do conteúdo da formatação e principalmente a simplicidade tanto para humanos quando para computadores. Assim nasceu o XML.

Para ter uma noção veja o conteúdo da **Listagem 1**. Ele é um conteúdo de um arquivo TXT utilizado para exportar dados de uma aplicação qualquer para outra. Olhando para ele você seria capaz de me dizer do que se tratam os dados que estão sendo exibidos? Acredito que não. Daí a "ineficiência" dos arquivos TXT neste ponto. Eles não são auto descritivos. Agora observe as mesmas informações sendo exibidas através de um arquivo XML na **Listagem 2**. Muito mais claro, muito mais simples e muito mais amigável. Por este e outros motivos, temos no padrão XML uma poderosa "ferramenta" de integração.

Listagem 1. Dados em arquivo TXT

```
00013315220904200900000010689788530924478O1
00013315220904200900000010689788530925550O1
00013315220904200900000010689788530926113O1
00013315220904200900000010699788530927295O1
00013315220904200900000010699788530927516O1
00013315220904200900000010699788530927523O2
```

Listagem 2. Dados em arquivo XML

```
<?XML version="1.0" encoding="UTF-8"?>
<Dados>
  <Lote codigo="0001331522">
    <Notas>
      <Nota numero="0000001068" Data="09042009">
        <Produto>
          <CodBarra>9788530924478</CodBarra>
          <Quantidade>01</Quantidade>
        </Produto>
   <Produto>
    <CodBarra>9788530925550</CodBarra>
    <Quantidade>01</Quantidade>
  </Produto>
  <Produto>
    <CodBarra>9788530926113</CodBarra>
    <Quantidade>01</Quantidade>
  </Produto>
 </Nota>
 <Nota numero="0000001069" Data="09042009">
  <Produto>
    <CodBarra>9788530927295</CodBarra>
    <Quantidade>01</Quantidade>
  </Produto>
  <Produto>
    <CodBarra>9788530927516</CodBarra>
    <Quantidade>01</Quantidade>
  </Produto>
  <Produto>
    <CodBarra>9788530927523</CodBarra>
    <Quantidade>02/Quantidade>
  </Produto>
 </Nota>
</Notas>
</Lote>
</Dados>
```

Como é de se observar o código da **Listagem 2** é muito mais fácil de ser compreendido em relação ao código da Listagem 1, apesar dos dois conterem a mesma informação. Um analista que vá desenvolver um sistema para importar o arquivo da **Listagem 1**, ficará refém de uma documentação que o diga onde começa e onde termina cada um dos campos, quais são e como estão formatados os tipos e demais informações que nos auxilie a fazer o parser correto do arquivo.

Já no caso da **Listagem 2**, vemos que o arquivo XML é auto descritivo, ou seja, cada nó além do seu conteúdo traz também a informação de quem ele é e como está estruturado. É como o arquivo falasse por si só. Vamos viajar um pouco e tentar "Ouvir" o que o arquivo diz.

Na **Linha 1** da **Listagem 2** ele está dizendo: "Olá, eu sou um arquivo XML da versão 1.0 e estou formatado com charset UTF-8 e suporto caracteres diacríticos".

Na **Linha 2** ele diz: "Você está entrando em um nó chamado dados". A **Linha 3** diz: "Aqui começa um lote com um código 0001331522". A **Linha 4** informa: "Este lote possui notas e estas notas podem estar descritas abaixo".

Já na **Linha 5** o arquivo nos diz: "Aqui está uma nota de número 0000001068 com data 09042009". A mensagem da **Linha 6** é: "Os dados a seguir são de um produto" e as **Linhas 7** e **8** nos dizem o que este produto tem e qual o seu valor. Neste caso cada produto tem um CodBarra e uma Quantidade com seus respectivos valores. A **Linha 9** nos diz: "Aqui se encerra os dados do produto" para que a partir da **Linha 10** um novo produto seja informado.

Você percebeu que o arquivo XML quase que "fala" com a gente. Este padrão de arquivo permitiu que fossem criados mecanismos eficientes de leitura e escrita, mecanismos estes que são suportados por inúmeras linguagens, e que mais a frente veremos em ação no PHP.

Estrutura de um XML

Para ser considerado válido, um arquivo XML deve seguir algumas regras definidas pelo W3C. Isso irá garantir que este arquivo seja manipulado independente de plataforma ou linguagem e permitindo assim que ele cumpra o seu papel. Todo arquivo XML deverá ser iniciado com uma instrução padrão. Ela deverá vir na primeira linha do arquivo e contendo sempre a versão utilizada do padrão e caso você for usar elementos da língua portuguesa, como acentos, cedilha, etc., deverá informar a codificação utilizada para os caracteres: encoding="ISO-8859-1". Aqui temos um exemplo da instrução que deverá iniciar um arquivo XML:

```
<?XML version= "1.0" encoding= "ISO-8859-1"?>.
```

Após esta instrução inicial, o arquivo deverá trazer o que chamamos de elemento root ou tag root. Este é o elemento principal do arquivo. Só deverá conter um por arquivo, e este elemento é quem conterá todos os outros elementos descendentes ou subordinados. Na **Listagem 2** o elemento root é representado pelo nó <Dados>. Sem este elemento principal os dados ficariam desorganizados, ou melhor, soltos dentro do arquivo. Abaixo do elemento root poderemos ter um ou vários elementos secundários, também chamados de campos, estes "campos" é quem conterão os dados que poderão ser lidos pelos mecanismos de parser.

Analisando a **Listagem 2**, você irá reparar que os elementos estão aninhados, ou seja, num XML todos os elementos devem estar propriamente aninhados uns dentro dos outros. Neste momento é muito importante ressaltar que em um arquivo XML todos os elementos devem ser fechados, todas as tags de abertura devem obrigatoriamente ter uma tag de fechamento, assim para cada <tag> deverá haver sem exceção uma </tag>. Esta regra também se aplica também ao elemento root.

Outro detalhe é que em um arquivo XML as marcações fazem distinção entre caracteres maiúsculos e minúsculos. Desta forma a expressão <tag>exemplo</tag> seria inválida para um documento XML. Um arquivo XML também trata os espaços em branco de maneira diferente de outras estruturas, preservando estes espaços em branco.

Outra maneira que temos de incluir informações através de um arquivo XML é adicionando atributos aos elementos presentes nos arquivos. Estes atributos devem ser inseridos utilizando o par Name/Value e sempre colocando os valores entre áspas. Na **Listagem 2** temos um exemplo de atributos no elemento Lote: código= "0001331522".

Analisando a estrutura de um arquivo XML ele nos lembra em muitos aspectos a estrutura de um documento HTML, mas observe a **Tabela 1**. Ela mostra as principais comparações entre documentos HTML e documentos XML.

HTML	XML
Tag Limitadas	Tag Ilimitadas
Podem ou não fechar as Tag`s	Todas as Tag`s devem ser fechadas
Ignora espaços em brancos	Considera espaços em branco

Tabela 1. Comparação de HTML e XML

DOM

Como foi dito anteriormente os arquivos XML obedecem a um padrão que é independente de tecnologia. Isso permitiu que se criassem mecanismos para realizar o parser de arquivos XML. Os dois principais métodos de leitura e escritas em arquivos XML são DOM e SAX. A maioria das linguagens disponíveis no mercado já possui classes que implementam esses mecanismos e no nosso caso iremos ver como o PHP trabalha com DOM.

Nota do DevMan

DOM (Document Object Model - Modelo de Objetos de Documentos) é uma especificação da W3C, independente de plataforma e linguagem, onde se pode dinamicamente alterar e editar a estrutura, conteúdo e estilo de um documento eletrônico. Ou seja, permitindo que o documento seja mais tarde processado e os resultados desse processamento sejam incorporados de volta no próprio documento. A API DOM oferece uma maneira padrão de se acessar os elementos de um documento, além de se poder trabalhar com cada um desses elementos separadamente, e por esses motivos criar páginas altamente dinâmicas.

Que utilidade teria um arquivo XML se ele tivesse que ser lido como um arquivo TXT? Para facilitar o acesso e manipulação de um documento XML, o W3C criou um conjunto de interfaces denominadas de DOM (Document Object Model) para que pudéssemos explorar um arquivo XML de forma eficiente. Mas o DOM não se restringe apenas a documentos XML. Pense em DOM como se fosse um conjunto de objetos e métodos específicos para manipulação e acesso de um arquivo estruturado em tags, como um documento HTML. Com DOM temos uma representação de um documento XML em uma estrutura em árvore carregada na memória. Com isso podemos navegar no documento em ambas as direções e acessar cada elemento deste documento de forma muito simples. Para ilustrar como o DOM trabalha observe o código da **Listagem 3**. Ele mostra um exemplo de um pequeno arquivo XML.

Listagem 3. Exemplo de XML Simple<?XML version="1.0"

```
encoding="ISO-8859-1"?>
<books>
 <book>
  <author>Rodrigo Carreiro</author>
  <title>Como usar XML no PHP</title>
  <publisher>DevMedia</publisher>
 </book>
 <book>
  <author>Adriano Santos</author>
  <title>WebService no PHP</title>
  <publisher>DevMedia</publisher>
 </book>
</books>
```

A **Figura 1** mostra como o documento XML da **Listagem 3** ficaria estruturado caso fosse carregado para a memória utilizando o DOM.

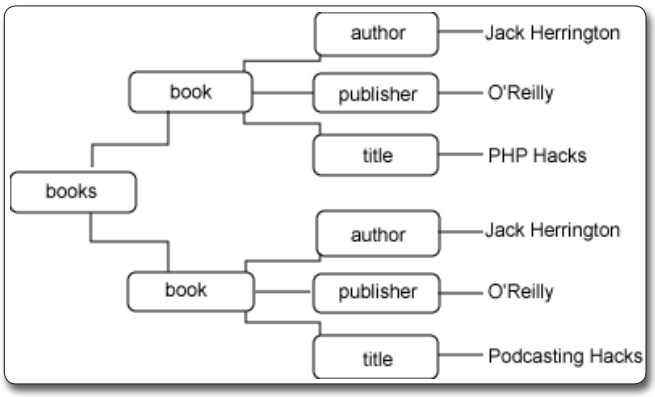


Figura 1. Árvore DOM do XML da Listagem 3

Agora vamos falar de código, vamos falar de programação, vamos falar de PHP. Como utilizamos o DOM dentro do PHP para, por exemplo, ler o documento XML exposto na **Listagem 3**? Muito simples. O PHP traz em sua composição, desde a versão 5, uma classe chamada DOMDocument que implementa as interfaces do DOM nos permitindo assim fazer um parser de um documento XML utilizando o DOM. Seu uso é bem intuitivo e a **Listagem 4** mostra um exemplo de código que utiliza DOMDocument para ler o arquivo XML da **Listagem 3**.

Nota

A classe DOM é nativa do PHP só a partir da versão 5 uma dica seria você instalar a biblioteca PHP_DOMXML.dll para manipular arquivos XML em versões anteriores do PHP.

Listagem 4. Codigo PHP utilizando DOM

```
<?PHP
$doc = new DOMDocument();
$doc->load('books.XML');

$books = $doc->getElementsByTagName()("book");
foreach($books as $book)
{
$authors = $book->getElementsByTagName()("author");
$author = $authors->item(0)->nodeValue;

$publishers = $book->getElementsByTagName()("publisher");
$publisher = $publishers->item(0)->nodeValue;

$titles = $book->getElementsByTagName()("title");
$title = $titles->item(0)->nodeValue;

echo "$title - $author - $publisher <br>";
}
?>
```

Observe o código da **Listagem 4**. A **Linha 1** é padrão, é a tag de abertura de um script PHP. A partir da **Linha 2** começamos nosso código. A primeira coisa é a criação de uma variável para receber a instância de uma classe DOMDocument. Ela será a responsável por permitir a navegação por todo o nosso documento e extrair dele as informações necessárias. Na **Linha 3** utilizamos o método load() para carregar do disco o conteúdo do arquivo books.xml. Neste momento também é feita uma validação para saber se este arquivo está dentro dos padrões que regem um arquivo XML. Uma exceção será gerada caso haja algo errado com o arquivo.

Uma vez tendo carregado o arquivo para a memória, nós extraímos todas as tags book presentes no documento através do método getElementsByTagName(). Este método retorna um array contendo as tags book com seus possíveis elementos. Na **Linha 4** temos a variável $books que receberá este array. Na **Linha 5** fazemos uma varredura neste array utilizando a função foreach do PHP e para cada item do array $books utilizamos novamente o método getElementsByTagName(), mas agora para extrairmos as informações da tag author. Como o método getElementsByTagName() retorna um array, a variável $authors na **Linha 7** recebe as informações da tag author do primeiro nó book. Na **Linha 8** apenas acessamos a primeira posição do array $authors e recuperamos o conteúdo da tag através do método nodeValue().

O processo descrito nas **Linhas 9**,**10**, **11**, **12** é o mesmo, porém extraímos informações das tags Publisher e Title. Para cada tag book encontrada dentro do loop nos invocamos a função echo do PHP e exibimos o conteúdo das variáveis no browser.

Acredito que você deva estar se perguntando: e se os nomes das tags mudarem? Não teria como obter um código genérico, que mapeasse os nós filhos independente de informamos o nome? Sim tem. E esta é a maneira mais eficiente de se mapear um arquivo XML, pois não fica restrito a uma estrutura específica. O código da **Listagem 5** mostra o exemplo de uma função que carrega um arquivo XML utilizando DOM, mapeia os nós e exibe o conteúdo no browser.

Listagem 5. Função para ler XML dinamicamente

```
function SearchXMLFile($filename, $noname)
{
  $DOM = new DOMDocument();
  $DOM->load($filename);

  $nos = $DOM->getElementsByTagName()($noname);

  foreach($nos as $no)
  {
    $childs = $no->childNodes;
    foreach($childs as $c)
   {
     Echo($c->nodeValue . "<br>");
   }
  }
}
```

Observe que no código da **Listagem 5** não fazemos mais referência explícita a nenhum nó ou elemento no XML. Nós iniciamos a função solicitando o caminho do arquivo XML que será lido, assim como o nome do nó pela qual começaremos a pesquisa. As **Linhas 3** e **4** mostram os códigos que se encarregam de criar um objeto DOMDocument e carregar o arquivo passado como parâmetro. Na **Linha 5** nós extraímos um array com todos os nós que possuem o nome passado como segundo parâmetro e para isso, usamos mais uma vez o método getElementsByTagName(). Na **Linha 6** fazemos um loop da mesma forma que no exemplo da **Listagem 4** a diferença fica por conta do código da **Linha 8**. Repare que agora nós utilizamos o método childNodes(). Ele irá recuperar os possíveis nós que por ventura venham existir. Feito isso mais um loop é realizado só que agora para extrair os valores propriamente ditos de cada nó.

Desta forma não ficamos presos a nomes de tags, apenas temos que informar o arquivo XML e o nome da tag pela qual iniciaremos a pesquisa. É claro que este código pode ser melhorado para que possamos mapear XML com infinitos níveis de aninhamentos. O princípio seria o mesmo, porém teríamos que fazer uma chamada recursiva a função. Mas isso vai ficar para outra oportunidade, pois agora vamos ver como usar o DOMDocument para gerar arquivos XML.

Criando XML com DOMDocument

Até aqui vimos o que é o DOM e como utilizá-lo dentro do PHP para poder ler arquivos XML. Agora, e se quisermos usar o DOM para, por exemplo, gerar um arquivo XML com dados vindos de uma ou mais tabelas do banco de dados? Isso seria possível e viável? Sim, a resposta é sim. Tão fácil quanto ler um arquivo XML é gerar um utilizando o DOM. Para mostrar na prática esta facilidade, vamos gerar um arquivo XML com dados dos países que temos em nosso planeta. Para isso utilizaremos um banco de dados chamado World que está disponível para o banco de dados MySql e pode ser baixado no site da própria MySql. Acesse http://downloads.mysql.com/docs/world.sql.zip e baixe o script para geração do banco de dados. Não entrarei em detalhes quanto a instalação do banco de dados bem como a execução do script, pois foge do escopo do nosso artigo, mas a nota do DevMan trás dicas sobre MySql.

ClubeDelphi PLUS!

Acesse agora o mesmo o portal do assinante ClubeDelphi e inscreva-se para o curso Rodrigo Carreiro que mostra como trabalhar com bancos de dados MySQL.
www.devmedia.com.br/cursos/listcurso.asp?curso=136

Este banco de dados possui dados de mais de 200 países e 4000 cidades ao redor do mundo. A idéia é criar uma função, utilizando DOMDocument, que gere um arquivo XML com os dados resultantes da instrução SQL passada para a mesma. O código da **Listagem 6** mostra com clareza como conseguiremos realizar este feito.

Listagem 6. Função para gerar um arquivo XML

```
function ExportToXML($select, $nameroot, $filename)
{
  $db = mysql_connect('localhost', 'root', 'root');
  mysql_select_db('dbworld');
  $res = mysql_query($select);

  $doc = new DOMDocument('1.0', 'UTF-8');
  $dados = $doc->createElement('Dados');
  $dados = $doc->appendChild($dados);

  while($row = mysql_fetch_assoc($res))
  {
     $reg = $doc->createElement($nameroot);
$reg = $dados->appendChild($reg);
foreach($row as $key => $value)
     {
  $field = $doc->createElement($key);
  $field = $reg->appendChild($field);
  $text = $doc->createTextNode($value);
  $text = $field->appendChild($text);
     }
     }
$doc->save($filename);
}
```

Tão fácil quanto ler um arquivo XML com DOM é gerar um. O código da **Listagem 6** mostra isso claramente. Observe que nossa função recebe três parâmetros que são: $select, aqui você deverá informar a instrução SQL que dará origem ao arquivo XML. O parâmetro $nameroot é o nome que você quer dar a cada registro do seu select, por exemplo, imagine que você queira gerar um XML com os dados de todas as cidades cadastradas no banco de dados, assim você poderia passar o valor 'cidade' para este parâmetro. Assim para cada cidade cadastrada no banco seria criada uma tag <cidade> e abaixo dela as tags de cada atributo da tabela cidade com seu respectivo valor. O terceiro parâmetro, $filename, é o nome do arquivo que você quer que seja gerado. Você poderá informar o caminho completo para o arquivo caso deseje.

Bem, já dentro da função, nas **Linhas 2**, **3**, e **4** temos os códigos necessários para se conectar a base de dados, selecionar o banco que queremos acessar e executar o comando SQL passado como parâmetro na nossa base de dados. Não entrarei em detalhes sobre essas funções por se tratarem de funções bastante conhecidas no meio da comunidade de desenvolvedores PHP, caso deseje mais informações sobre as mesmas acesse o manual do PHP on-line através do endereço: http://www.PHP.net/manual/pt_BR/ref.mysql.PHP

Na **Linha 5**, criamos nosso objeto DOMDocument de uma forma um pouco diferente. Agora informamos a versão e a codificação de caracteres utilizada. Esta é uma das maneiras que temos de passar estas informações para o objeto para que ele gere a tag raiz de forma correta. Como vimos no inicio deste artigo, todo arquivo XML deve possuir um elemento raiz, ao qual, todos os outros elementos estarão subordinados. É isto que fazemos na **Linha 6**. Através do objeto DOMDocument, representado pela variável $doc, nós criamos nosso elemento root. A função createElement sempre retorna o ponteiro do elemento que acabou de ser criado. Nós guardamos este ponteiro na variável $dados, pois como um documento XML deve estar estruturado de forma hierárquica, temos que ir adicionando os nós filhos debaixo do nível do nó pai, e começamos a fazer isso na **Linha 7** através do método appendChild().

Criado o elemento root e já tendo o ponteiro para o nó subordinado a ele, na **Linha 8** iniciamos um laço para percorrer o ResultSet armazenado na variável $res. A função mysql_fetch_assoc() transforma cada linha deste resultset em um array associativo, ou seja, para cada linha do resultset ela irá criar um array associativo onde a chave é o nome da coluna, é o valor do conteúdo desta coluna. Nas **Linhas 9** e **10** nós criamos uma tag para a linha do resultset em que estamos parados e adicionamos esta tag debaixo do nó anterior (**Linha 10**). Repare que esta tag recebe como Caption o valor passado via parâmetro pela função. Feito isso, temos que percorrer todas as colunas desta linha em que estamos e para cada coluna criarmos uma tag e inserirmos nela o valor do campo correspondente no banco de dados. As **Linhas 11** a **15** ilustram como isso é feito.

Ao final dos dois laços temos em memória a estrutura do nosso arquivo XML pronta para ser gerada no disco. Na **Linha 16** apenas invocamos o método save() do objeto DOMDocument e informamos o caminho onde desejamos que o arquivo seja gerado. Feito isso basta invocar a função passando para a mesma os parâmetros necessários para a geração do arquivo. A **Figura 2** mostra o conteúdo de um arquivo XML que foi gerado a partir da chamada da função com os seguintes parâmetros:

```
function ExportToXML("Select * from Country","Pais","C:\pais.XML");
```

Conclusão

O DOM é apenas uma das bibliotecas ou extensões presentes no PHP 5 para se trabalhar com XML. Temos ainda o SAX que em comparação ao DOM se mostra mais eficiente quando manipula arquivos muito grandes. Porém, a implementação do SAX, em minha opinião é um tanto confusa visto que ele trabalha com chamadas a funções definidas pelo usuário para responder a determinados eventos que ocorrem durante o parser de um arquivo XML, como início ou fim de uma tag.

Bom seria se tivemos uma biblioteca que unisse o melhor dos

Figura 2. Arquivo XML gerado com DOM

dois mundos, a facilidade do DOM com a eficiência do SAX. E tem. Ela se chama SimpleXML e está habilitada por padrão a partir do PHP 5. É a maneira mais fácil de trabalhar com XML. Não é preciso decorar a extensa API do DOM. Com ela basta acessar o XML através de uma representação da estrutura de dados. Mas isso é assunto para outra oportunidade.

Quer com DOM, SAX ou SimpleXML o fato é que se você ainda não precisou de uma delas, com certeza um dia vai precisar. Então não perca tempo! Ponha em prática os conceitos aprendidos aqui e aumente sua lista de experiências acrescentando XML com PHP. Grande abraço. Eu sou Rodrigo Carreiro e pela sua atenção muito obrigado. ●

Dê seu feedback sobre esta edição!

A Clubedelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Existem momentos onde a estrutura faz toda a diferença.
Se o tempo é ruim, você encontra na Alog um porto seguro. Isto porque sua empresa reduz custos e melhora a governança em TI. Entre em contato e descubra porque a Alog é líder em hosting gerenciado.
REDUÇÃO DOS CUSTOS
AUMENTO DA GOVERNANÇA EM TI
guadalupe
0800 282 3330 | alog.com.br
ALOG
DATA CENTERS DO BRASIL
HOSTING GERENCIADO | COLOCATION | CONTINGÊNCIA | E-MAIL CORPORATIVO | CONECTIVIDADE | SERVIÇOS PROFISSIONAIS

VOCÊ levou anos para desenvolver, testar e colocar seu produto no mercado e **ELE** apenas alguns minutos para **PIRATEÁ-LO.**

SafeNet tem à sua disposição a proteção de software mais avançada e segura. As Chaves de Hardware Sentinel oferecem tecnologia que incorpora criptografia de chave pública, criptografia AES e autenticação interna para garantir o mais alto nível de proteção anti-pirataria.

Peça seu Kit de Desenvolvimento hoje mesmo.

Mais informações www.safenet-inc.com/antipirataria/03
e receba o Estudo de Caso.

SafeNet®
The Foundation of Information Security

***Sr. Cliente favor ler as informações com atenção, dúvidas entrar em contato com atendimento.**

Relatório de Pré-Flight

OP: 70059 Cliente: Neoficio Data: 17/08/09 Operador:JULIO

Informações da OP:
Título: Clob Delphi
Quantidade de Páginas: 48 + Capa
Formato do Miolo: 205x275
Formato da Capa: 205x275
Tamanho da Lombada: ---------
Tipo de Acabamento: Canoa

Informações do Arquivo:
Título: Clob Delphi
Quantidade de Páginas: 48 + Capa
Formato do Miolo: 205x275
Formato da Capa: 205x275
Tamanho da Lombada: ---------
Tipo de Acabamento: Canoa

Informações do Arquivo

• Imagens em baixa resolução com menos de 200 DPI:
Páginas: Todo o Miolo
Menos de 100 DPI: Capa 02
• Texto preto em quatro cores nas páginas:
Páginas:
• Imagens em RGB nas páginas:
Páginas:

• Colofon / Tiragem: ( ) OK ( ) Não
• Código de Barras: ( ) OK ( ) Não
• Páginas sem sangria e/ou inferior a 3 mm (Filete branco no acabamento):
Páginas: *Capa-03*
• Texto rente ao corte nas páginas:
Páginas:

OBS: As informações em **VERMELHO** necessitam de atenção, se de acordo favor assinar.

Data ____/____/____ Cliente: _____

Nome Legível

Todas as Provas de Cor devem ser assinadas de acordo com a aprovação. Havendo alteração na cor favor anotar, e será enviado nova Prova de Cor.

Observações: